To develop a library of speech processing algorithms based on fixed point format.

Gursewak Singh Sidhu

To develop a library of speech processing algorithms based on fixed point formats.

Report submitted to
Indian Institute of Technology, Kharagpur
for the award of degree

of

Bachelor of Technology
in Electrical Engineering

by

Gursewak Singh Sidhu

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR
May, 2014

CERTIFICATE

This is to certify that the Dissertation Report entitled, "**To develop a library of speech processing algorithms based on fixed point formats**," submitted by **Mr. Gursewak Singh Sidhu** to Indian Institute of Technology, Kharagpur, India, is a record of bonafide Project work, carried out by him under my supervision and guidance in the Department of Electrical Engineering.

_____

Prof. Aurobinda Routray

DECLARATION

I certify that

a. the work contained in this report is original and has been done by me under the guidance of my supervisor, Prof. Aurobinda Routray.
b. the work has not been submitted to any other Institute for any degree or diploma.
c. I have followed the guidelines provided by the Institute in preparing the report.
d. I have conformed to the norms and guidelines given in the Ethical Code of conduct of the Institute.
e. Wherever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

_____

Signature of the Student

ACKNOWLEDGEMENT

I take the opportunity to express my reverence to my supervisor Prof. Aurobinda Routray for his guidance, inspiration and innovative technical discussion during the course of this project. His perpetual energy and enthusiasm in research has motivated us. In addition, he was always accessible and willing to help in this project.

I would also like to thank Mr. Bibek Kabi for his support, who always found time between his busy research schedules to help me when stuck at any problem.

I would also like to extend my gratitude to all the professors for their contribution in my studies and research work. They have been great source of inspiration for me.

Last but not the least I would like to thank my parents, who taught the value of hard work by their own example.

Gursewak Singh Sidhu,

Roll no.: 11EE35013

Department of Electrical Engineering

IIT Kharagpur

Abstract

In majority of commercially available processors on the market today there is no hardware support for floating-point arithmetic due to cost the extra silicone imposes on a processor's total cost. This necessitates software emulation for the floating point arithmetic. This software overhead can significantly limit the rate at which algorithms can be executed.

By implementing algorithms using fixed-point mathematics, a significant improvement in execution speed can be observed because of the inherent integer math hardware support in a large number of processors. This speed does come at the cost of reduced range and accuracy of the algorithms variables. With increased use of battery operated portable systems, the need for long battery life can be supported by specialized systems optimized for that particular application in fixed point format. The purpose of this project is to build a fixed point library for speech processing algorithms.

In the first phase of BTP most of the time has been dedicated to studying, understanding and implementing speech processing algorithms in floating point in C++. The error analysis was done using Matlab fixed point support.

CONTENTS

Chapter: 1

INTRODUCTION

To accurately construct an algorithm, double or single precision floating-point data and coefficients values should be used. However there is significant processor overhead required to perform floating-point calculations resulting from the lack of hardware based floating-point support. In some cases such as in lower powered embedded processors there is not even compiler support for double precision floating point numbers. Floating-point overhead limits the effective iteration rate of an algorithm.

To improve mathematical support throughout or increase the execution rate, calculations can be performed using two's complement signed fixed-point representation. Fixed-point representation requires the programmer to create a virtual decimal place in between two bit locations for a given length of data. Fixed point implementation results in increased speed of the algorithm but it comes at the cost of range and accuracy. The motive of the project is to develop a fixed point DSP library for speech algorithms that uses optimal word length of fixed point.

Q format explained

Q is a fixed point number format where the number of fractional bits is specified. Q format is often used in hardware that does not have a floating point unit and in applications that require constant resolution. Q format numbers are fixed point numbers that is, they are stored and operated upon as regular binary numbers (i.e. signed numbers), thus allowing standard integer hardware/ALU to perform rational number calculations. The number of integer bits, fractional bits and the underlying word length are to be chosen by the programmer on an application-specific basis. The notation used for Q format representation in this paper is Qm.n, where

Q designates that the number is in Q format notation. (m) is the number of bits set aside to designate the two's complement integer portion of the number, exclusive or inclusive of the sign bit. (n) is the number of bits used to designate the fractional portion of the number, i.e. the number of bits to the right of the binary point.

In this report the varying Qm.n format haven been used for different algorithms and the final result are represented in Q5.26 format where 32 represents the word length, 26 represents the number of decimal point bits , and 5 represents integer part of result.

DIGITAL MODELS FOR SPEECH

In order to apply digital signal processing techniques to speech processing problems it is essential to understand the fundamentals of the speech production process. Speech signals are composed of sequence of sounds. These sounds and the transition between them serve as a symbolic representation of the information.

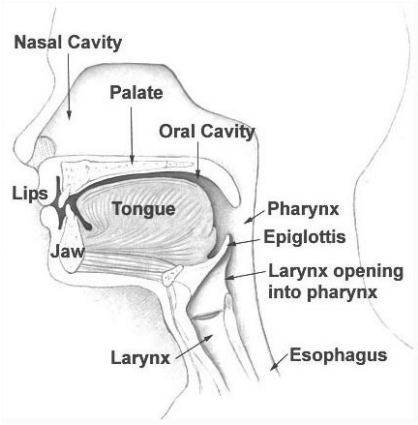


Fig:1 Human Vocal Tract

The human vocal tract consists of the pharynx and the mouth or oral cavity. In average male, the total length of vocal tract is 17cm. The cross-sectional area of vocal tract, determined by the positions of the tongue, lips, jaw, and velum varies from zero to about $20cm^2$. The nasal tract begins at the velum and ends at the nostrils. When the velum is lowered, the nasal tract is acoustically coupled to the vocal tract to produce the nasal sounds of speech. The sub-glottal system, consisting of lungs, bronchi and trachea serves as a source of energy for the production of speech. Speech is simply acoustic wave that is radiated from this system when air is expelled from the lungs and the resulting flow of air is perturbed by a constriction somewhere in the vocal tract.

Speech signals can be classified in three distinct classes according to mode of excitation.

- **Voiced sounds**: are produced by forcing air through the glottis with the tension of the vocal cords adjusted so that they vibrate in a relaxation oscillation, thereby producing quasi-periodic pulses of air which excite the vocal tract.
- **Unvoiced sounds:** are generated by forming a constriction at some point in the vocal tract and forcing air through the constriction at a high enough velocity to produce turbulence. This creates a broad spectrum noise source to excite the vocal tract.
- **Plosive sounds**: result from making a complete closure, building up pressure behind the closure and abruptly releasing it.

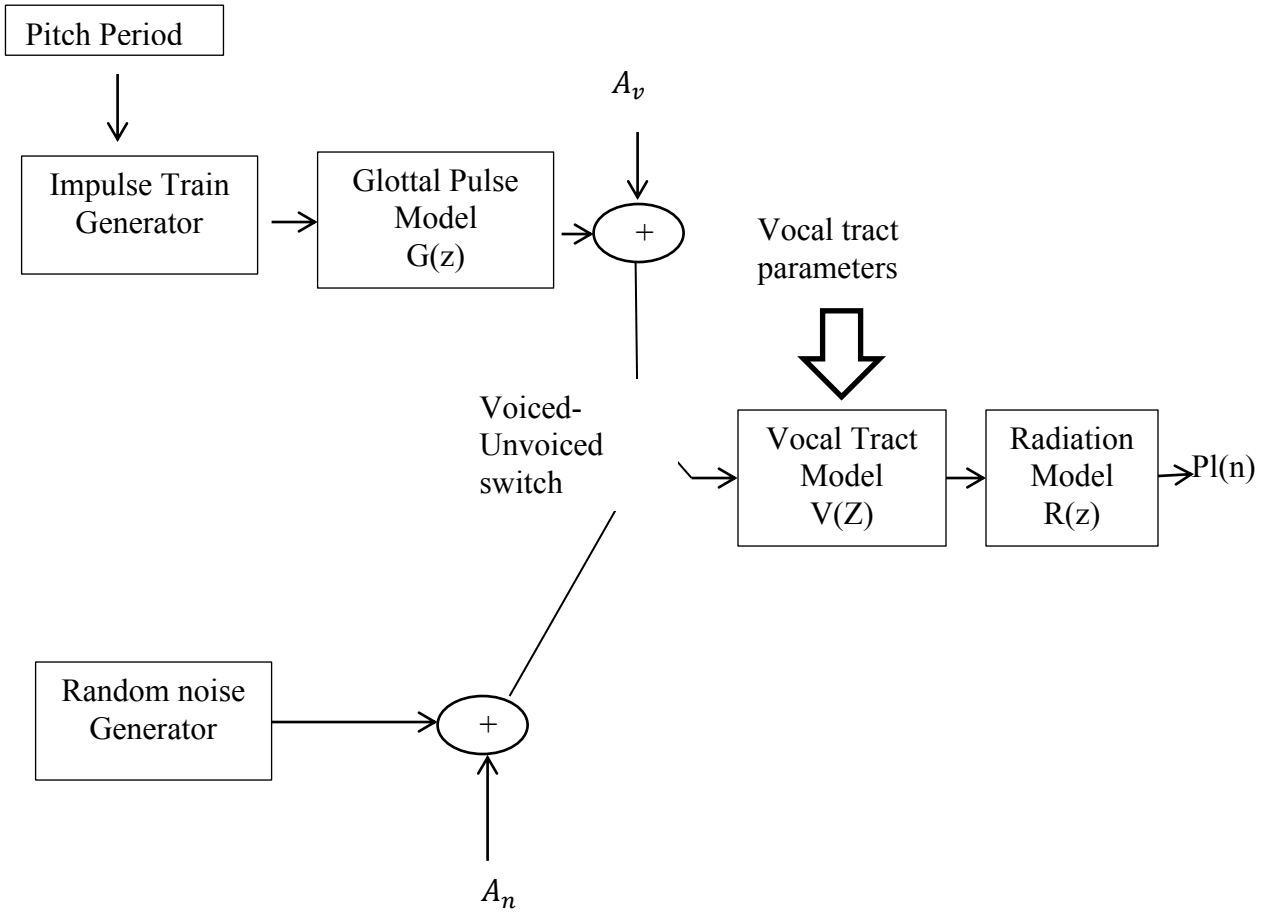
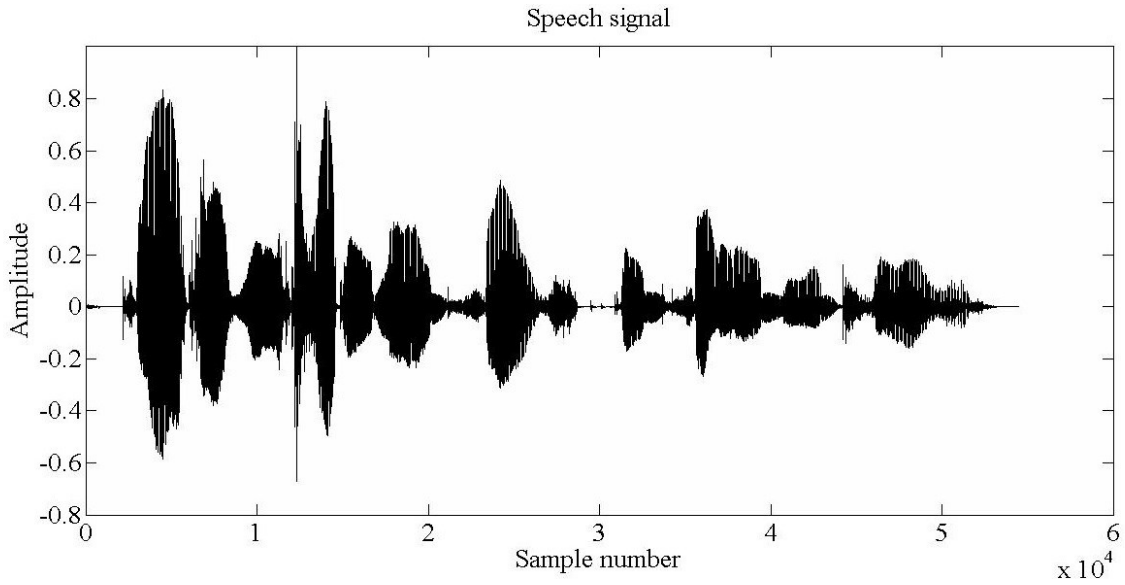


Fig 2. General discrete time model for speech production

Time Dependent processing of speech



A sequence of samples representing a typical speech signal is shown.

It is evident from the figure that the properties of the speech signal change with time. For example, the excitation changes between voiced and unvoiced speech, there is significant variation in peak amplitude of the signal and fundamental frequency.

The underlying assumption in most speech processing schemes is that the properties of the speech signal change relatively slowly with time. This assumption leads to a variety of "short time" processing methods in which short segments of speech are isolated and processed as if they were short segments from a sustained sound with fixed properties. This is repeated as often as desired. Often these short segments, which are called analysis frames, overlap each other. The result of processing on each frame may be either a single number or a set of numbers. Therefore such processing produces a new time-dependent sequence which can serve as a representation of speech signal.

Chapter: 2

Algorithms Explained

1) **Short time Energy:** The amplitude of unvoiced segments is generally lower than the amplitude of voiced segments. The short time energy of a speech signal provides a convenient representation that reflects these amplitudes variations. The short time energy is defined as

$$E_n = \sum_{m=-\infty}^{\infty} [x(m)w(n-m)]^2$$

The choice of window function determines the nature of the short-time energy representation. The effect of windows is illustrated by using two windows

a) Rectangular window

b) Hamming window

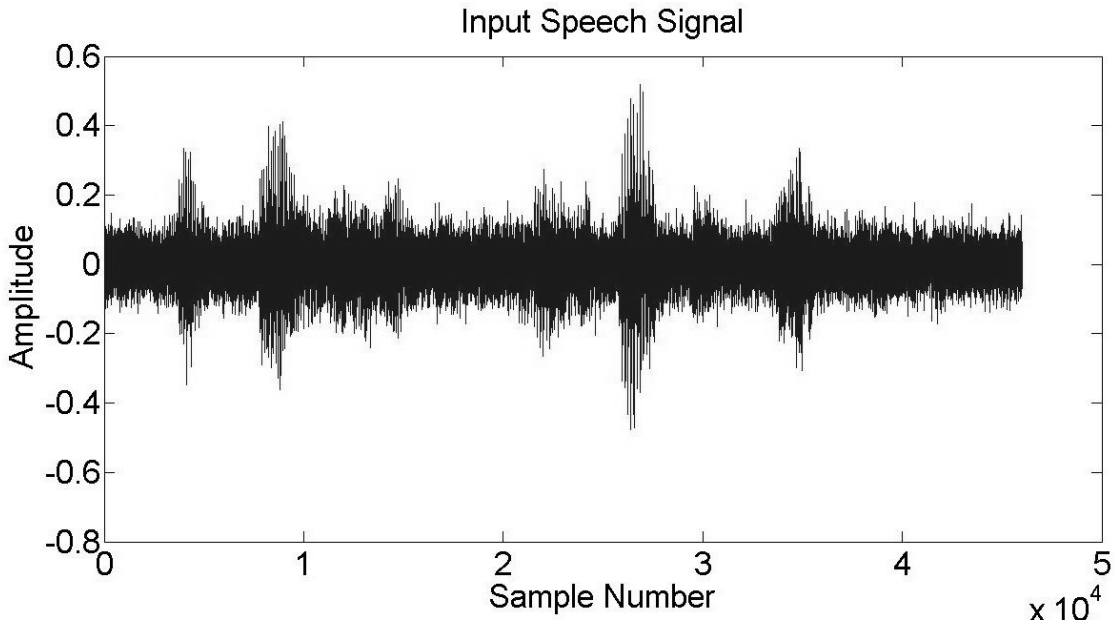


Fig 1.a

The major significance of short time energy calculation is that it provides a basis for distinguishing voiced speech segments from unvoiced speech segments. As can be seen from the plots the values of E_n are quite small for unvoiced speech segments and this can be exploited to approximately locate the unvoiced speech segments from voiced speech segments. Q5.26. Figure 1.a shows the input speech signal, figure 1.b shows the average

absolute energy calculations in floating point format and figure 1.c shows the same calculations in fixed point format. The mean error observed was of order 10^{-9}. The window size taken was 10ms with 0% overlap.

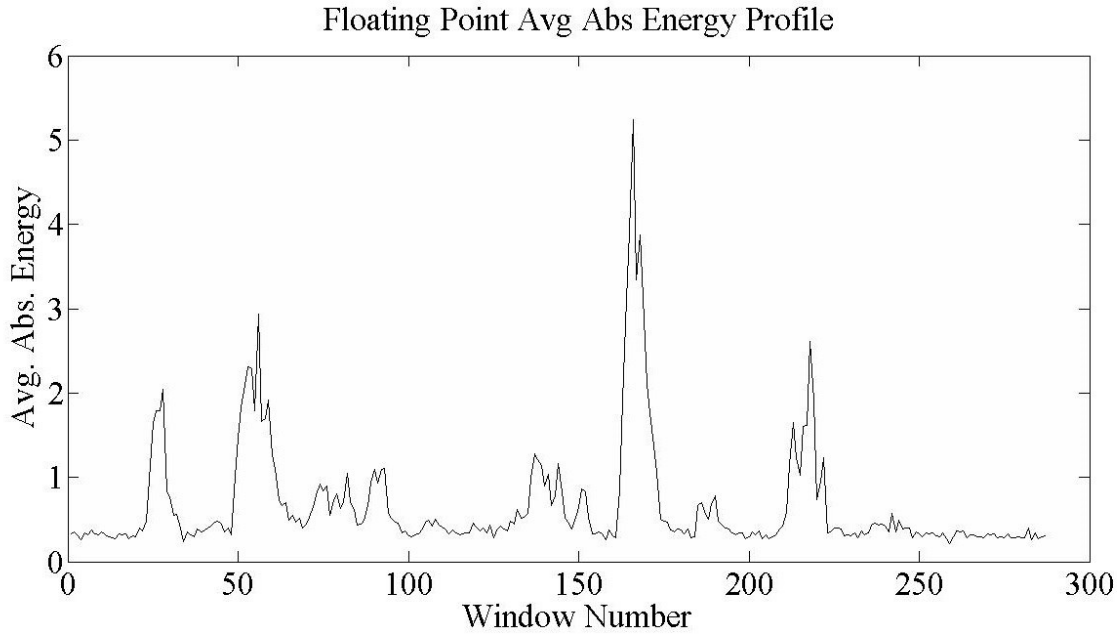


Fig 1.b

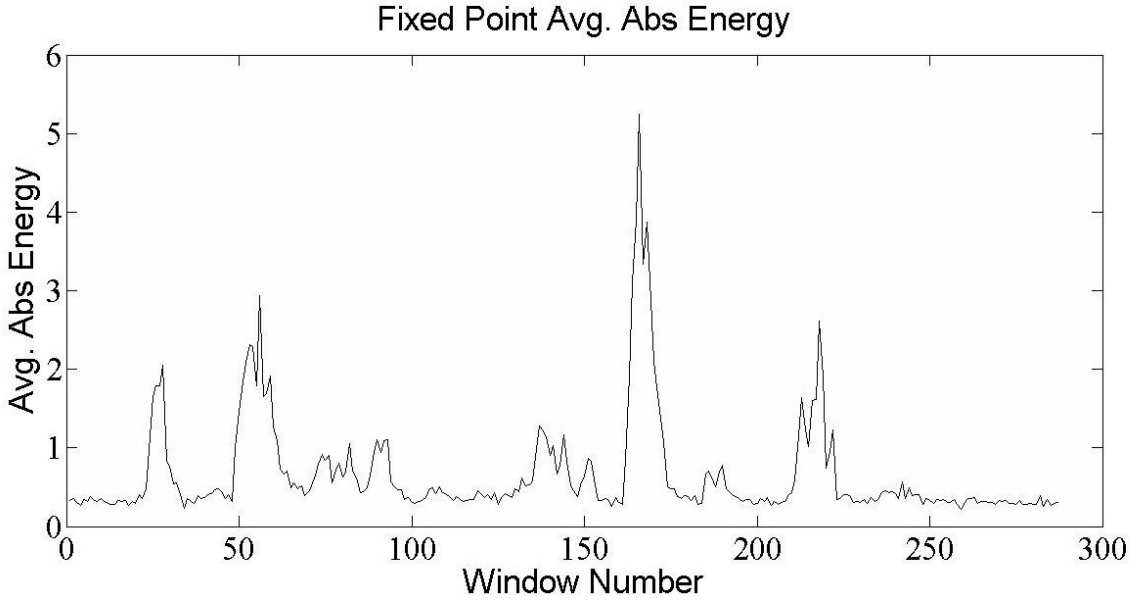


Fig 1.c

Algorithm 1:

Input: Speech signal, window size (in ms), Sampling Frequency
Output: Average absolute Energy of provided window

Function Average_absolute_Energy (-------, -------)
{

a. Break the input speech signal in provided window size. Let N denote the number of samples in each window

N = window_size * Sampling_frequency / 1000

b. Calculate energy as

Energy = 0; // initialize the energy variable

For i=1:1:N
{
Energy = Energy + x(i) * x(i);
}

c. Return (Energy)

}

If sampling frequency is quite high (say more than 8 kHz) then the number of samples in each window will be much larger which shall result in higher energy value and thus we shall require more integer bits to represent it. Hence it would be better to first down sample the speech signal to 8 KHz.

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0.5
Avg. Energy	<32,6,25>	4.9599	8.8466×10^{-6}	0.3466

2) **Short time Average Magnitude:** A problem with short-time energy function is that it is very sensitive to large signal values, ($E_n \propto [x(n) * x(n)]^2$) which emphasizes a lot on sample to sample variation of x(n). Short time Average magnitude function elevates this problem as

$$\boldsymbol{M_n = \sum_{m=-\infty}^{\infty} |x(m)|w(n-m)}$$

where the weighted sum of absolute values of a signal is computed rather than the sum of squares.

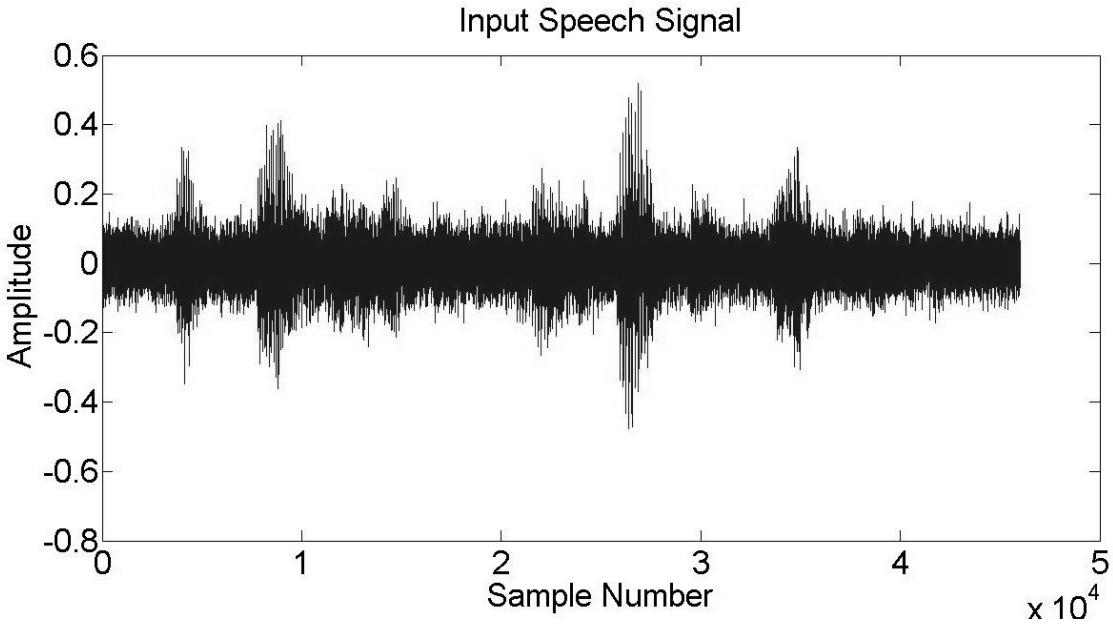


Fig 2.a

The other window functions could also be utilized and there is no restriction on use of any window function. The average magnitude function is also utilized to distinguish between voiced and un-voiced segment. Q5.26 format was used for above simulation. Mean error observed was in order of 10^{-9}. Fig 2.a shows the plot of a input speech signal. The window size taken was 10ms with 0% overlap.

Fig 2.b shows average absolute amplitude calculated using floating point format and Fig 2.c shows the calculations being done in fixed point format.

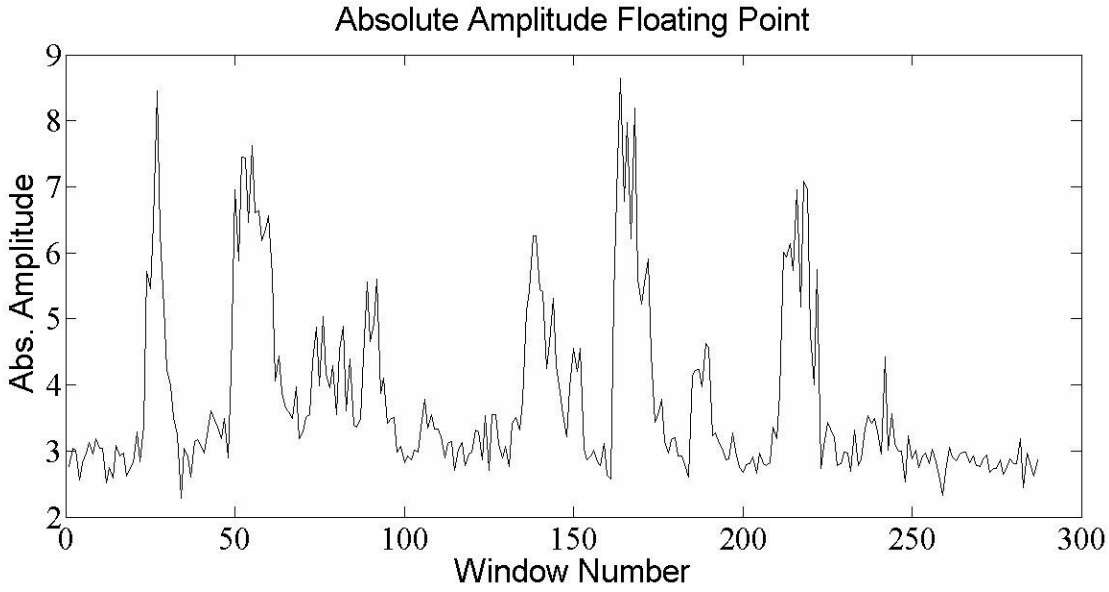


Fig 2.b

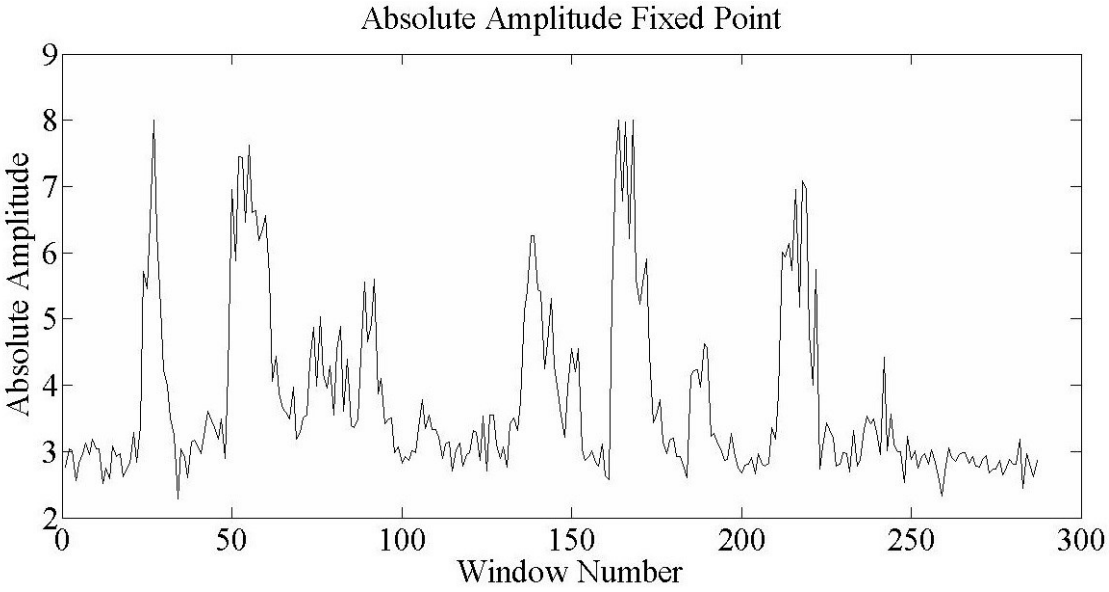


Fig 2.c

Algorithm 2:

Input: Speech signal, window size (in ms), Sampling Frequency
Output: Average absolute Amplitude of provided window

Function Average_absolute_Amplitude (-------, -------)
{

a. Break the input speech signal in provided window size. Let N denote the number of samples in each window

N = window_size * Sampling_frequency / 1000

b. Calculate Abs_Amplitude as

Abs_Amplitude = 0; *// initialize the Abs_Amplitude variable*

For i=1:1:N
{

Abs_Amplitude = Abs_Amplitude + abs (x(i));
}

c. Return (Abs_Amplitude)

}

If sampling frequency is quite high (say more than 8 kHz) then the number of samples in each window will be much larger which shall result in higher Abs_Amplitude value and thus we shall require more integer bits to represent it. Hence it would be better to first down sample the speech signal to 8 KHz.

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0.5
Avg. abs Magnitude	<32,6,25>	20.8534	0.0295	4.1538

3) **Short time Average Zero-Crossing Rate:** A zero crossing defines the number of times the sign of signal changes within a specified window. The rate at which zero crossing occurs is simply a measure of frequency content of the signal. This is particularly true for narrowband signals with zero dc bias. Speech signals are broadband signals and hence interpretation of zero-crossing rate is therefore much less precise. However, rough estimates of spectral properties can be obtained for it. The definition of Zero Crossing rate is

$$Z_n = \sum_{m=-\infty}^{\infty} |sgn(x(m) - sgn(x(m-1))|w(n-m)$$

Where sgn[x(n)] = 1 x(n) >0

= -1 x(n) <0

w(n) is a windowing function

The speech is mostly concentrated below 4 KHz, whereas most of the noise if of high frequency. Since, high frequency applies high zero crossing rates and low frequency applies low zero crossing rates, there is a strong co-relation between the zero crossing rate and speech segment being voiced or unvoiced. If zero crossing rates are high, then the window is pure noise and if low, the given window is speech window. Fig 3.a shows the input speech signal on which calculations were carried out. The Q5.26 format was used for fixed point simulations. Fig 3.b shows the normalized Zero crossing rate in floating point and Fig 3.c shows the Zero crossing rate in fixed point format. Mean error observed was of order10^{-8}.

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0.5
Avg. zero cross rate	<32,6,25>	20.8534	0.0295	4.1538
Temp	<32,31,0>	202	0	64

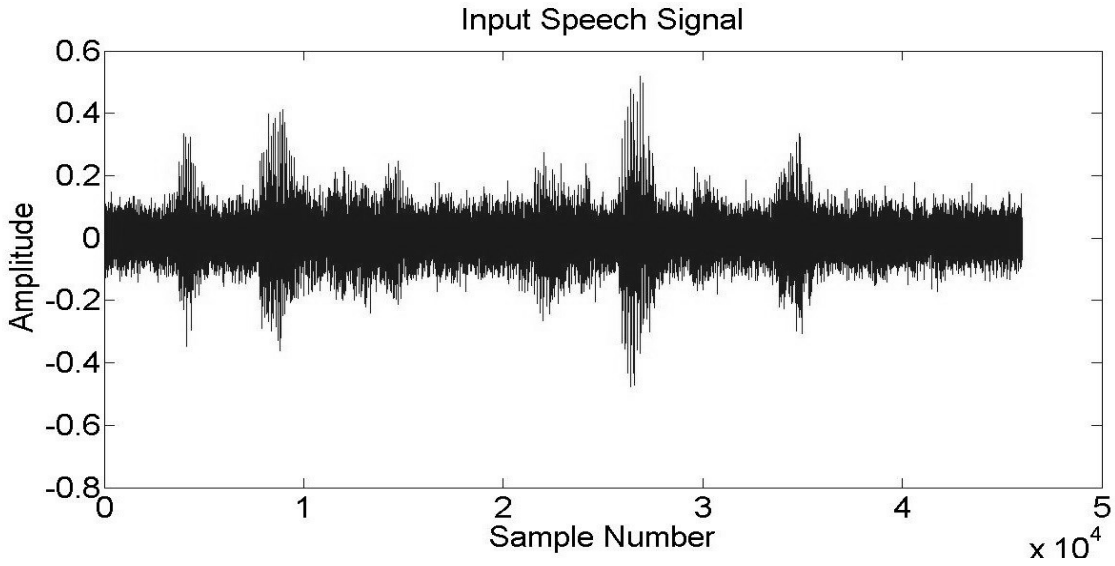


Fig 3.a

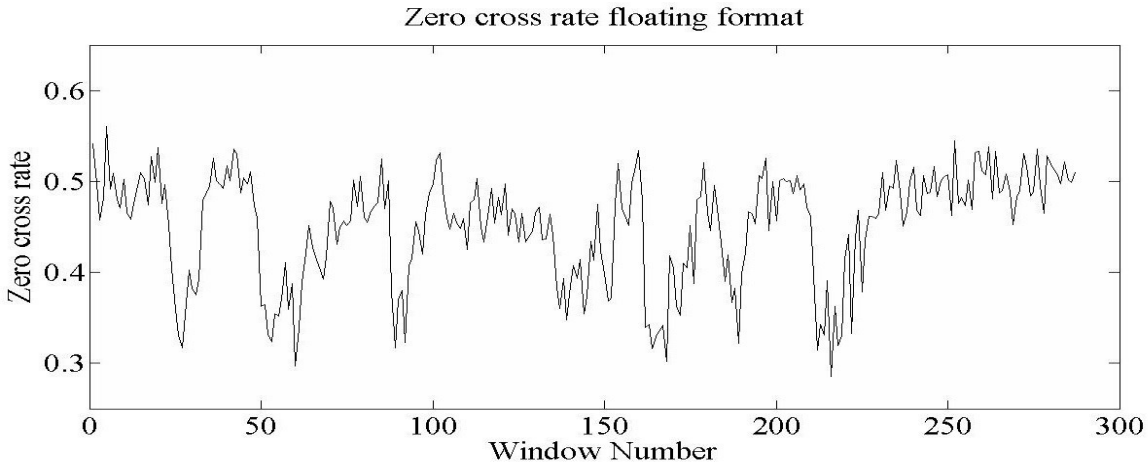


Fig 3.b

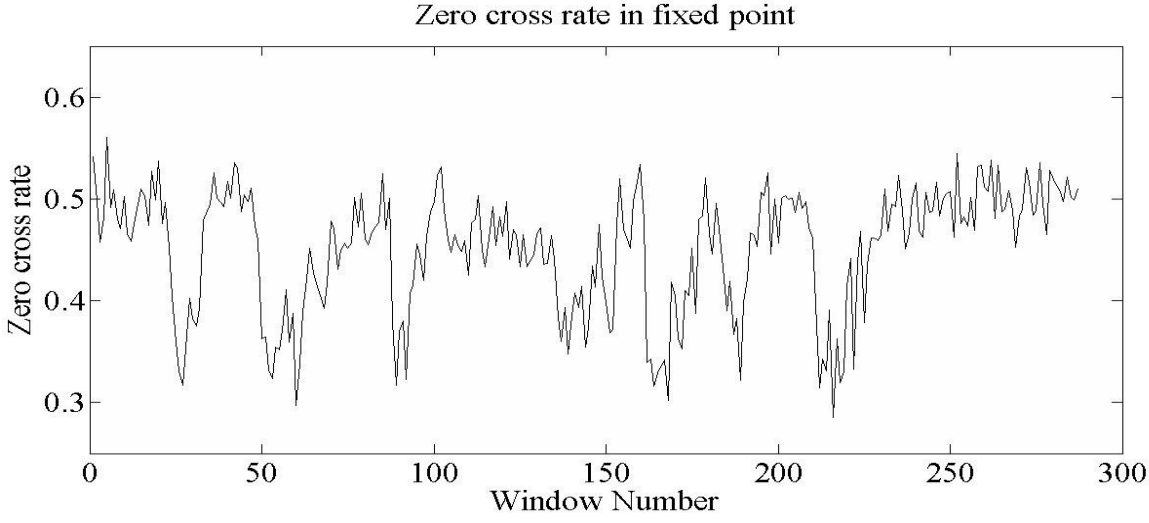


Fig 3.c

Algorithm 3:

Input: Speech signal, window size (in ms), Sampling Frequency
Output: Zero Cross Rate of provided window

Function Zero_cross_rate (-------, -------)
{

a. Break the input speech signal in provided window size. Let N denote the number of samples in each window

N = window_size * Sampling_frequency / 1000

b. Calculate zero_cross_rate as

zero_cross_rate = 0*; // initialize the zero_cross_rate variable*

For i=1:1:N
{

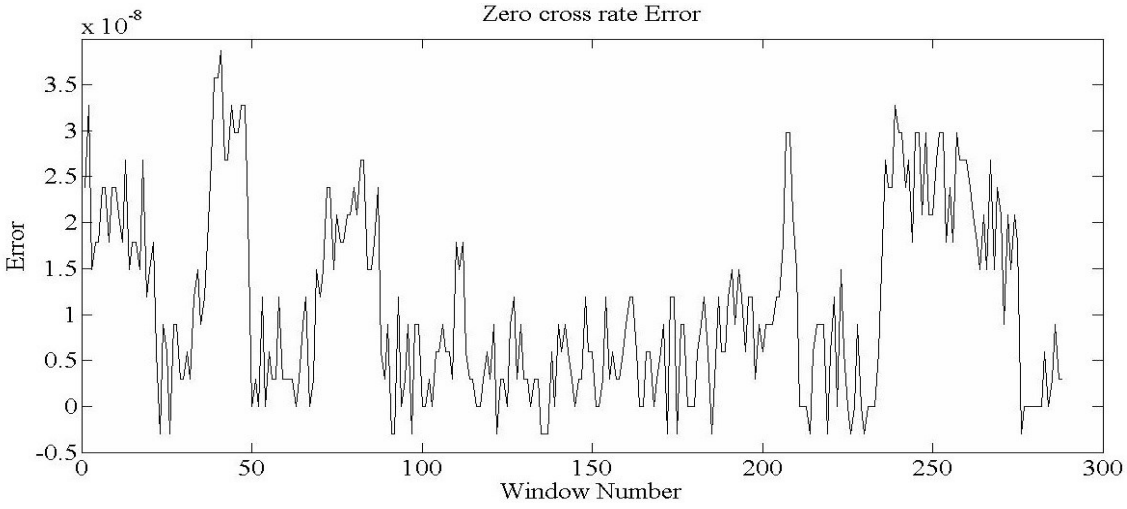
zero_cross_rate = zero_cross_rate + $|sgn(x(m) - sgn(x(m-1))|$

}

c. Return (zero_cross_rate)

}

If sampling frequency is quite high (say more than 8 kHz) then the number of samples in each window will be much larger which shall result in higher calculations. Down sampling won't affect the zero cross rate. Hence it would be better to first down sample the speech signal to 8 KHz.



Error of zero cross rate floating point vs fixed point

4) **Short time Autocorrelation function:** Autocorrelation of a discrete time deterministic signal is defined as

$$\varphi(k) = \sum_{m=-\infty}^{\infty} x(m)x(m+k)$$

Autocorrelation of a periodic signal is also periodic with the same period. Some other properties of autocorrelation function are

- It is even function
- It attains maximum at value of k=0
- The autocorrelation at k=0 is equal to energy of deterministic signal or power of a random or periodic signal.

Short time autocorrelation function is defined as

$$R_n(k) = \sum_{m=-\infty}^{\infty} x(m)w(n-m)x(m+k)w(n-k-m)$$

This implies that first a window of speech is selected by applying a window function and then the deterministic autocorrelation is applied to the windowed speech segment.

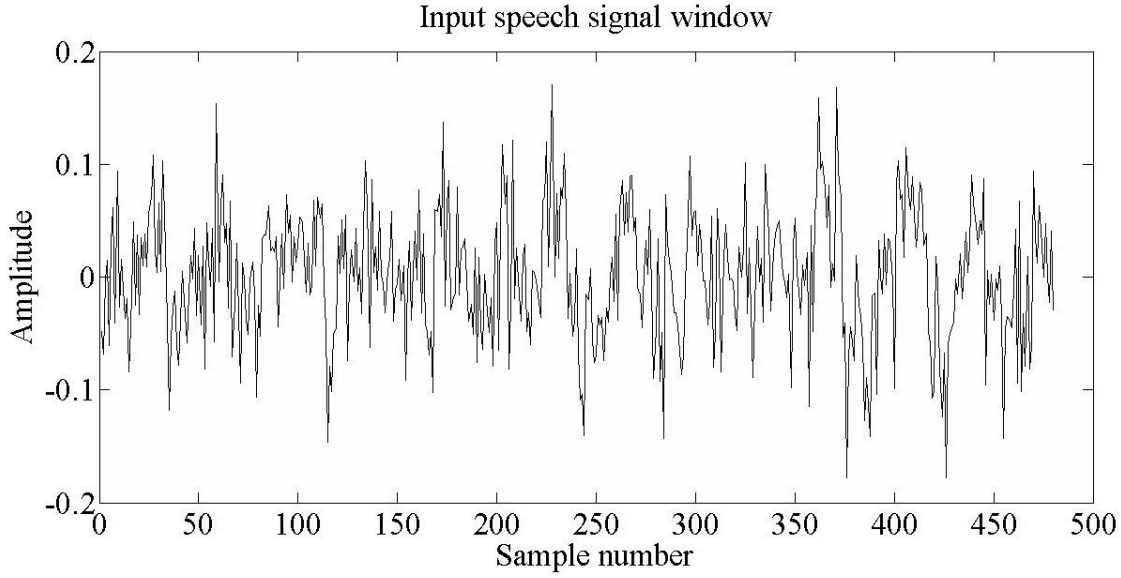


Fig 4.a

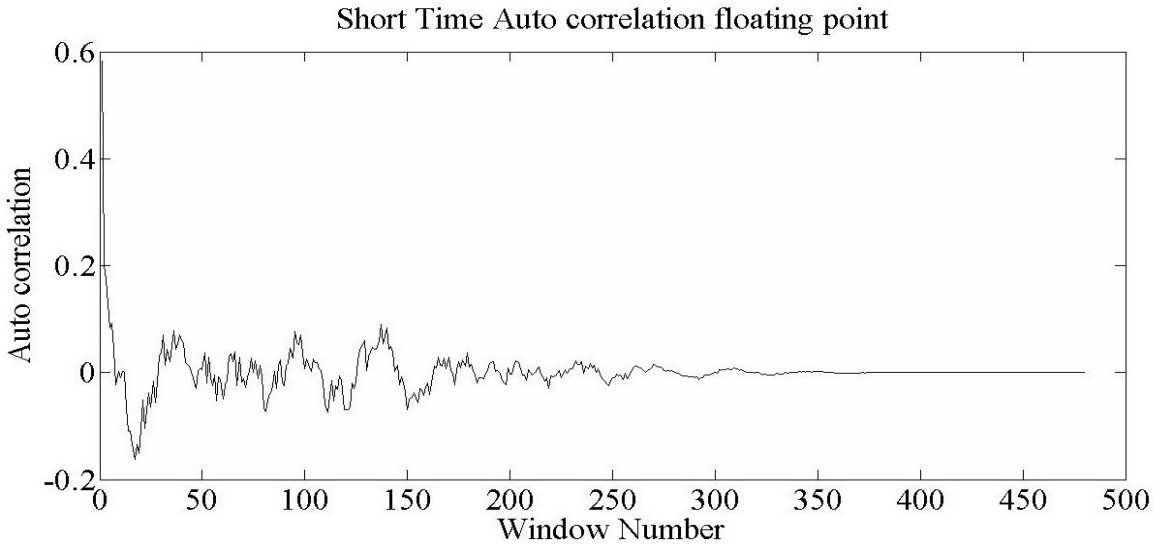


Fig 4.b

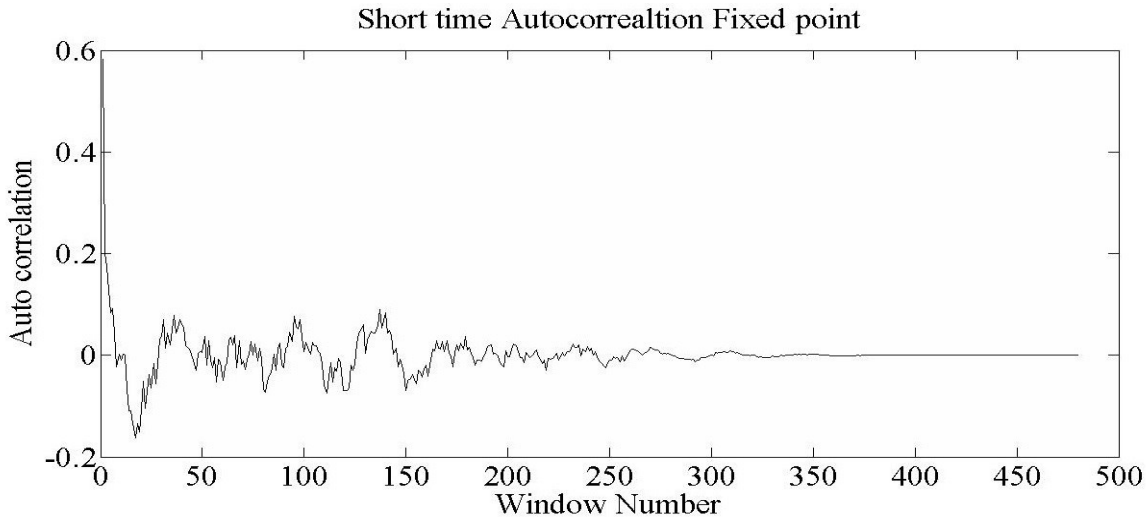


Fig 4.c

Fig 4.a shows the input speech signal window on which calculations were performed. The problem with autocorrelation function is that it is very calculation intensive and hence mostly not preferred when alternatives do exist. Fig 4.b shows the Short Time Auto Correlation function calculation for floating point format anf Fig 4.c for fixed point format of Q5.26. The mean error observed was of order 10^{-10}.

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0.5
Auto-correlation	<32,6,25>	4.9598	-3.1544	0.0014

Algorithm 4:

Input: Speech signal, window size (in ms), Sampling Frequency
Output: Autocorrelation of provided window

Function Autocorrelation (-------, -------)
{

a. Break the input speech signal in provided window size. Let N denote the number of samples in each window

N = window_size * Sampling_frequency / 1000

b. Calculate autocorrelation as

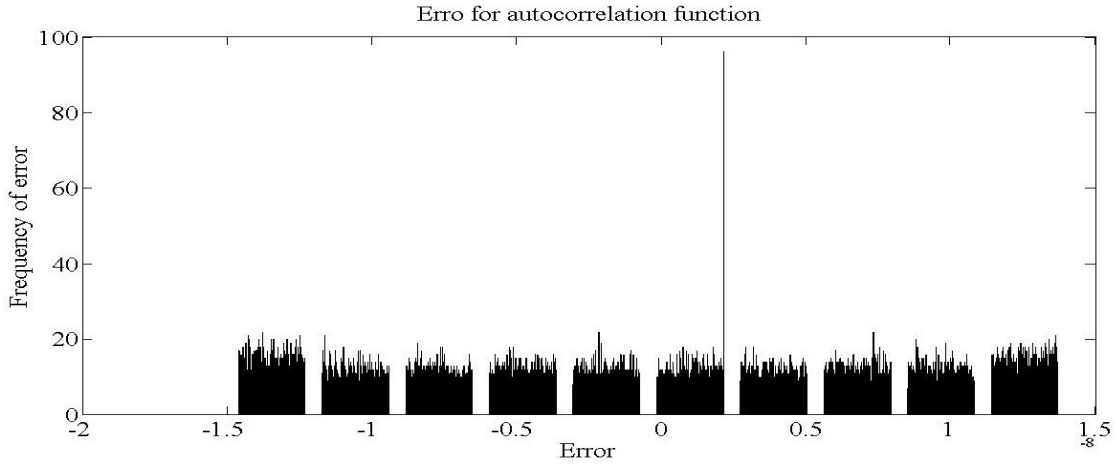
For k=1:1:N
{
autocorrelation(k) = 0;

for m=1:1:N-k
{
autocorrelation (k) = autocorrelation (k) + $x(m) * x(m+k)$;
}
}

c. Return (zero_cross_rate)

}

If sampling frequency is quite high (say more than 8 kHz) then the number of samples in each window will be much larger which shall result in higher calculations. Hence it would be better to first down sample the speech signal to 8 KHz. The down sampling won't affect the final envelope of the autocorrelation function.



Error for autocorrelation function

5) **Speech vs. Silence Discrimination Using Energy and Zero-Crossings:** The problem of locating the beginning and end of a speech utterance in a background noise is of importance in many areas of speech processing. A scheme for locating the beginning and end of a speech signal can be used to eliminate significant computation in nonreal-time systems by making it possible to process only the parts of the input that correspond to speech. The algorithm given below works on the principle that first 100msec of speech data is pure noise. Even if the above case is not true then for most of real time applications we have a separate mike to record the noise profile after a fixed interval and that can be utilized effectively with no change in algorithm.

Table 5.1	Window 1	Window 2	Window 3	Window 4	Window 5	Window 6
Floating point speech windows	21-33	38-99	126-157	162-182	185-198	208-227
Fixed point speech windows	21-33	38-99	126-157	162-182	185-198	208-227

Fig 5.a shows the input speech signal used for test purpose. Fig 5.b shows the Matlab results for fixed point and floating point. No error was observed for 15 cases that were tested. Table 5.1 shows the windowed speech segments which are voiced. Q5.26 format was used for test purposes.

The algorithm can be quite useful when high rate of processing is required ex. for speech recognition as the calculations can be reduced by at least one thirds for normal speaker. The other use of this algorithm is noise removal. Most noise removal algorithms require a sample of noise after a fixed interval to update their references for noise removal. This algorithm was used for above mentioned purpose in SVD based speech enhancement and Spectral subtraction method, as explained later in report.

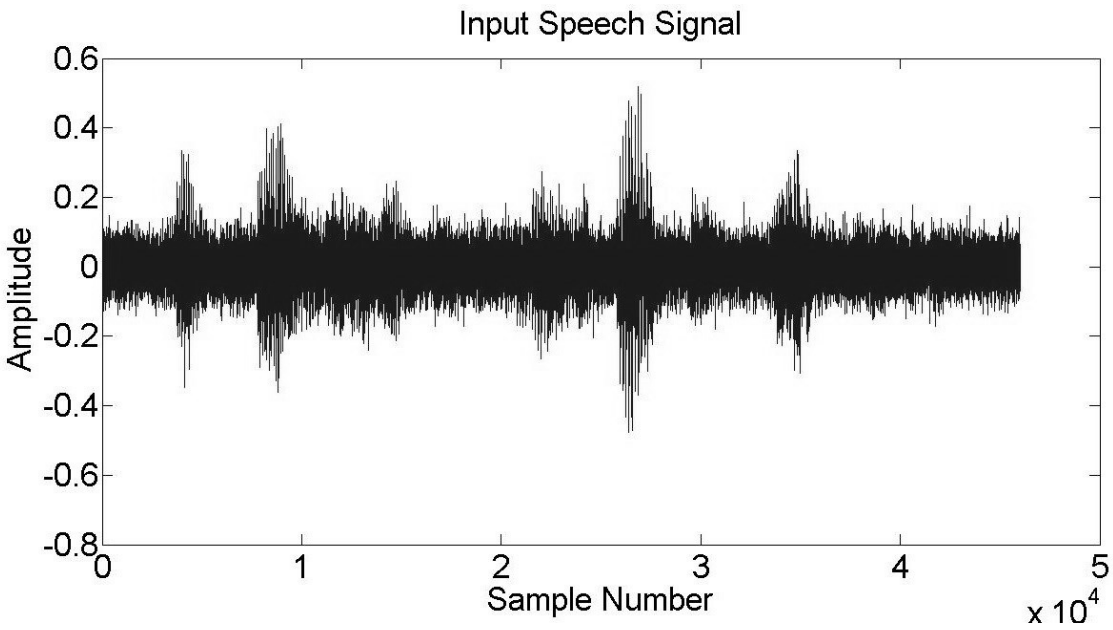

Input Speech Signal
Amplitude
Sample Number
x 10^4
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
1
2
3
4
5

Fig 5.a

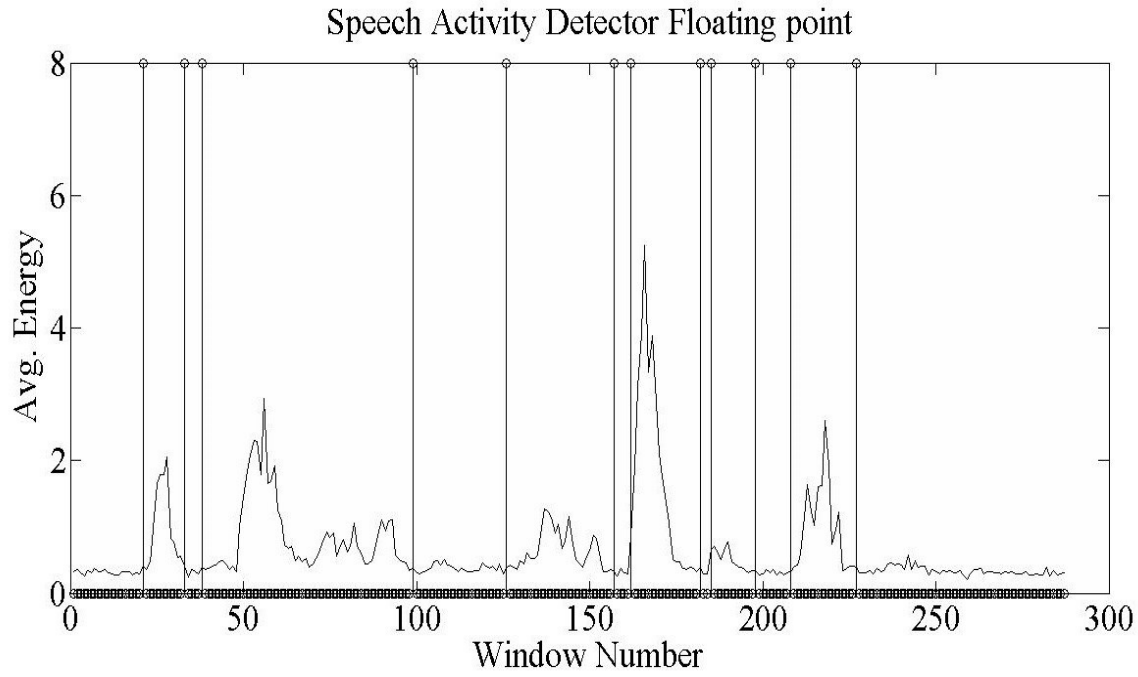

Speech Activity Detector Floating point
Avg. Energy
Window Number
8
6
4
2
0
0
50
100
150
200
250
300

Fig 5.b

Algorithm 5:

Input: Speech signal, window size (in ms), Sampling Frequency
Output: defines whether given window is voiced or not voiced

Function Activity_Detector (-------, -------)
{

a. Break the input speech signal in provided window size. Let N denote the number of samples in each window

N = window_size * Sampling_frequency / 1000

b. Calculate avg_abs_energy using algorithm 1.
c. Calculate avg_abs_amp using algorithm 2.
d. Calculate zero_cross_rate using algorithm 3.

e. Let no of windows = no_windows

no_windows = size_of_input_speech_signal / N;

Here we have considered zero overlapping between windows.

```
Noise_avg_energy = 0;
Noise_abs_amplitude = 0;
Noise_zero_cross_rate =0;

// Calculating window values for noise. The assumption is first //
100ms of speech signal is pure noise.
For k=1:1:10
{
    Noise_avg_energy= Noise_avg_energy + avg_abs_energy(k)
    Noise_abs_amp = Noise_abs_amp + avg_abs_amp(k) ;
    Noise_zero_cross_rate = Noise_zero_cross_rate +
                                zero_cross_rate;
    }

// Assume a threshold value of 1.5 – 2 times of Noise_abs_amp
i = 1;
flag =0;
For k=1:1:no_of_windows
{
    If ( avg_abs_amp (k) > Noise_abs_amp *1.5 )
            {
                N(i) = k;
                k = k+1;
            While ( avg_abs_amp (k) > Noise_abs_amp *1.5 )
                {
                    k = k+1;
                }
                M(i) = k;
                i = i+1;
                }
        }
```

```
// N and M provides the basic start and end points of the signal
// Now we shall use average energy to get another approximation
No_of_segments = i-1;
i = 1;
For k=1:1:no_of_segments
{
  If ( zero_cross_rate (N1(k)) > Noise_zero_cross_rate & N1(k) <=N(k) )
        {
            N1(i) = N1(k);
            k = k+1;
            }
Else
        {
            N1(k) = N1(k)+1;
            }
If ( zero_cross_rate (M(k)) < Noise_zero_cross_rate & M(k) <=M1(k) )
        {
            M(i) = M(k);
            k = k+1;
            }
Else
        {
            M(k) = M(k)+1;
            }
  }
// Now N1(i) and M(i) defines the start and end of a voiced segment of
speech. If the noise is varying the values of Noise_abs_amp,
Noise_abs_energy and Noise_zero_cross_rate should be updated on
regular intervals. If sampling is above 8 KHz , its better to downsample the
signal to 8KHz as it reduces the amount of calculations and have no effect
on above algorithm.
```

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0.5
Avg. Energy	<32,6,25>	4.9599	8.8466×10^{-6}	0.3466
Avg. abs Magnitude	<32,6,25>	20.8534	0.0295	4.1538
Avg. zero cross rate	<32,6,25>	20.8534	0.0295	4.1538

6) **Pitch period estimation using Autocorrelation function:** Autocorrelation function has many peaks and most of these peaks can be attributed to the damped oscillations of the vocal tract response which are responsible for the shape of each period of speech wave. Formant frequencies can also cause a problem as their peak might be higher than the pitch period peak. Hence, to avoid above problems we need some pre-processing of the speech signals. This process is known as spectrum flatteners. Here we shall discuss an algorithm to detect pitch of a speech using autocorrelation method based on clipping. Various steps of the algorithm are

Test pitch signal	Fixed Point output	Percent error
100 Hz	100	0 %
250 Hz	250.5682	0.2273 %
440 Hz	441	0.2273%

Table 6.1

In Fig. 6.a , the speech signal for a window is being plotted. In Fig 6.b the plot shows the speech signal with clipped version and in Fig 6.c the new altered signal, on which autocorrelation is performed to calculate the pitch. Standard pitch 100 Hz, 250 Hz and 440 Hz signal was used to the performance of the signal.

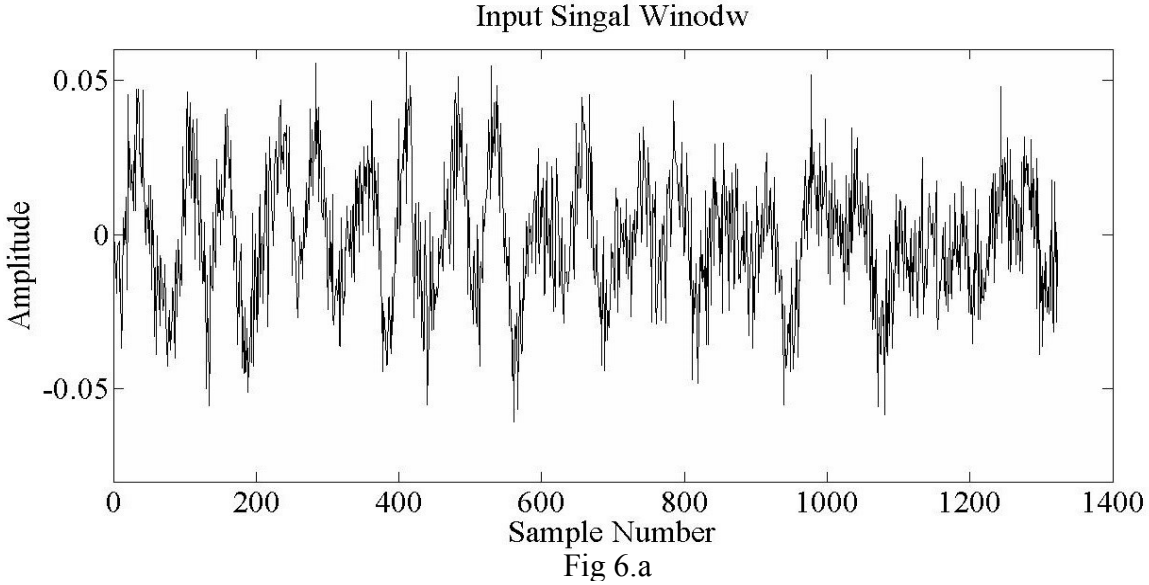


Fig 6.a

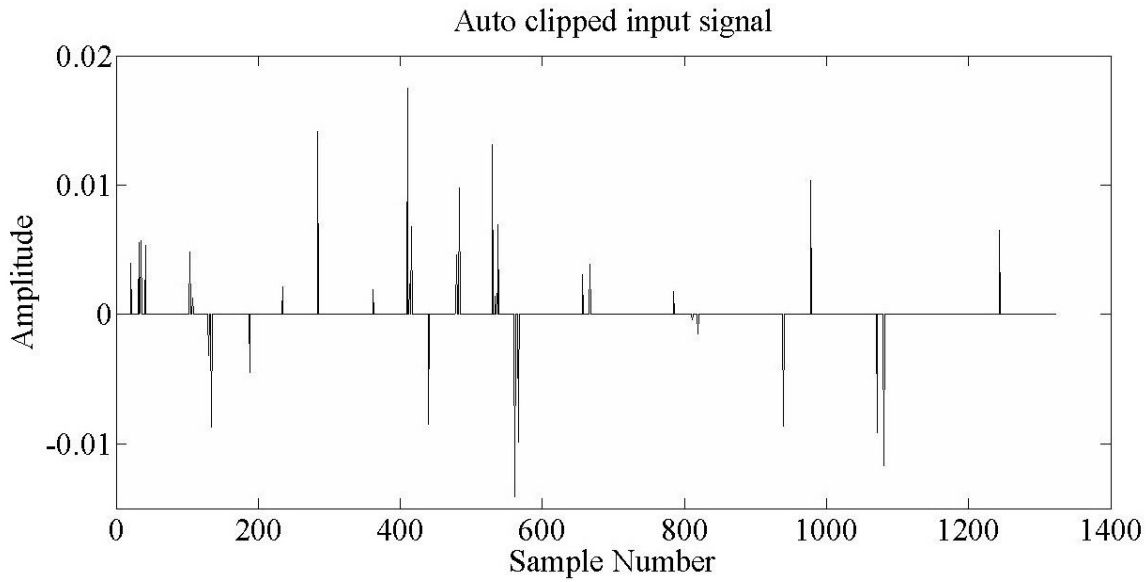

Auto clipped input signal
0.02
0.01
0
-0.01
Amplitude
0
200
400
600
800
1000
1200
1400
Sample Number

Fig 6.b

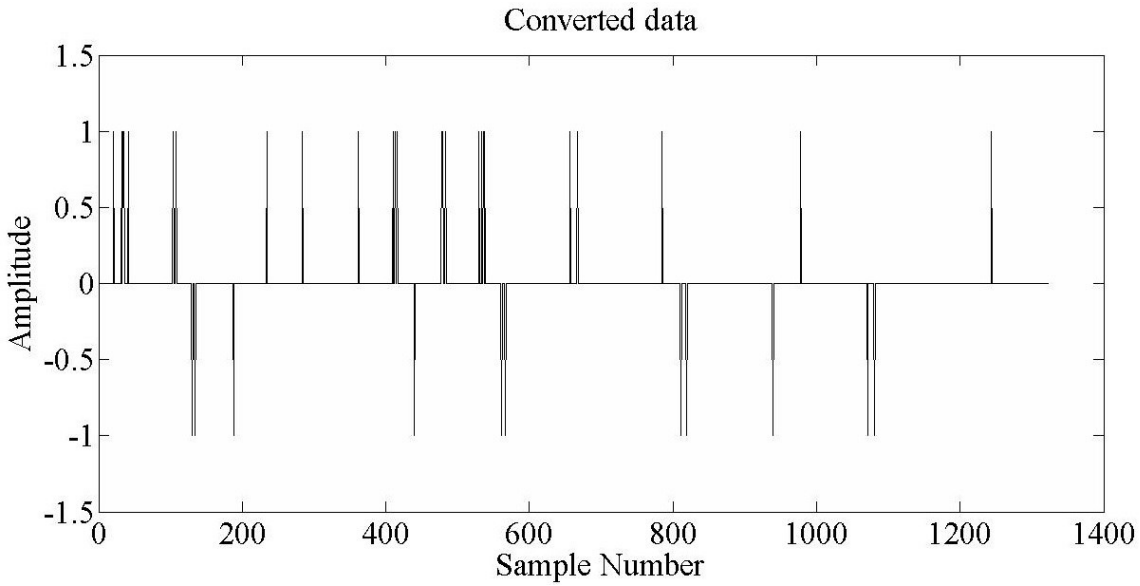

Converted data
1.5
1
0.5
0
-0.5
-1
-1.5
Amplitude
0
200
400
600
800
1000
1200
1400
Sample Number

Fig 6.c

Algorithm 6:

Input: Speech signal, window size (in ms), Sampling Frequency
Output: pitch of the window

Function Pitch_estimator(-------, -------)
{

a. Spectrally flatten the input speech signal using LPC
b. Pass the spectrally flattened speech signal through a low pass filter with cut off frequency 900Hz, so that formant frequencies don't come in picture.
c. Break the input speech signal in 30ms with an overlap of 50% between successive windows. Let N denote the number of samples in each window

N = 30 * Sampling_frequency / 1000

```
// For a windowed data
Max_value = 0;
Min_value = 0;
For i=1:1:N
        {
                If (x(i) > max_value)
                        {
                                Max_value = x(i);
                                }
                Else if (x(i) < min_value)
                        {
                                Min_value = x(i);
                                }
                }
Upper_peak_value = 0.8 * Max_value;
Lower_peak_value = 0.8* Min_value;

// Clipping function
For i=1:1:N
        {
                If (x(i) > Upper_peak_value)
                        {
                                x(i) = 1;
                                }
                Else if (x(i) < Lower_peak_value)
                        {
                                x(i) = -1;
                                }
                Else
                        {
                                x(i) = 0;
                                }
```

```
                                        }
                                }
// Find autocorrelation of the new signal using algorithm 4.

// Peak Detector for pitch estimation other i=0 to i= 12 as human pitch frequency lies between 60 to 600 Hz normally considering the signal is sampled at 8KHz

Peak_value = 0;
I_value = 0;

For i=12:1:N
        {
                If (autocorrelation (i)  > Peak_value)
                        {
                                Peak_value = autocorrelation (i) ;
                                I_value = I;
                                }
                }

If ( I_value = 0)
{
        Return (0);
        }
Else
        {
                Return (8000/I_value);
                }
        }
```

If sampling frequency is more than 8KHz, the signal should be down sampled to 8KHz as it reduces the amount of calculations. Another way could be to further down sample the signal, find the maximum value of autocorrelation function, then calculate the autocorrelation value around the I_value in input signal to fine tune the value. These way calculations can be reduced to a lot extent. The algorithm can be easily implemented in hardware as to find autocorrelation we could just use a counter as the input signal is only 1, 0 and -1.

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0
Processed speech	<32,1,30>	1	−1	0
Auto-correlation	<32,31,0>	480	0	240

7) **Median Smoothing and speech processing:** Linear smoother is used to eliminate noise like components in a signal. But linear smoother is not ideal for all applications, in case of pitch detection as it removes signal discontinuities. Hence we need a filter which can remove the large errors as well as retain discontinuities in the signal. Median filter followed by linear smoothening can be utilized in such cases. The given algorithm in the library provides smoothening by utilizing median filtering and linear smoothening.

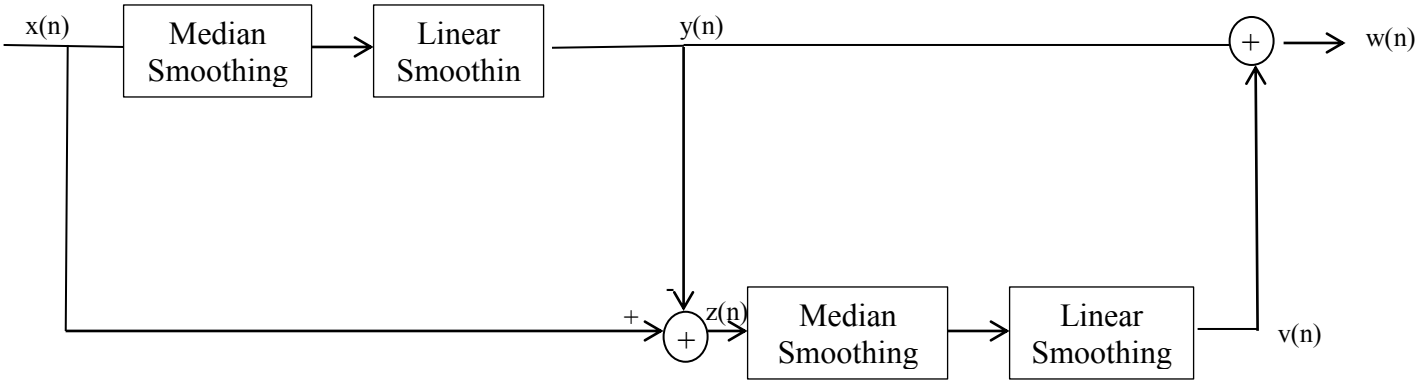


Fig 7.a

For Linear smoothing Hanning filter with impulse response

h(n) = ¼ n=0
= ½ n=1
= ¼ n=2

In above block diagram x(n) is speech signal taken as input. Here y(n) is smoothened part of x(n). z(n) is rough part of x(n) and v(n) represents the smoothened part of z(n).

$w(n) = S[x(n)] + S[R[x(n)]]$

where S denotes the smoothening function

R denotes rough part.

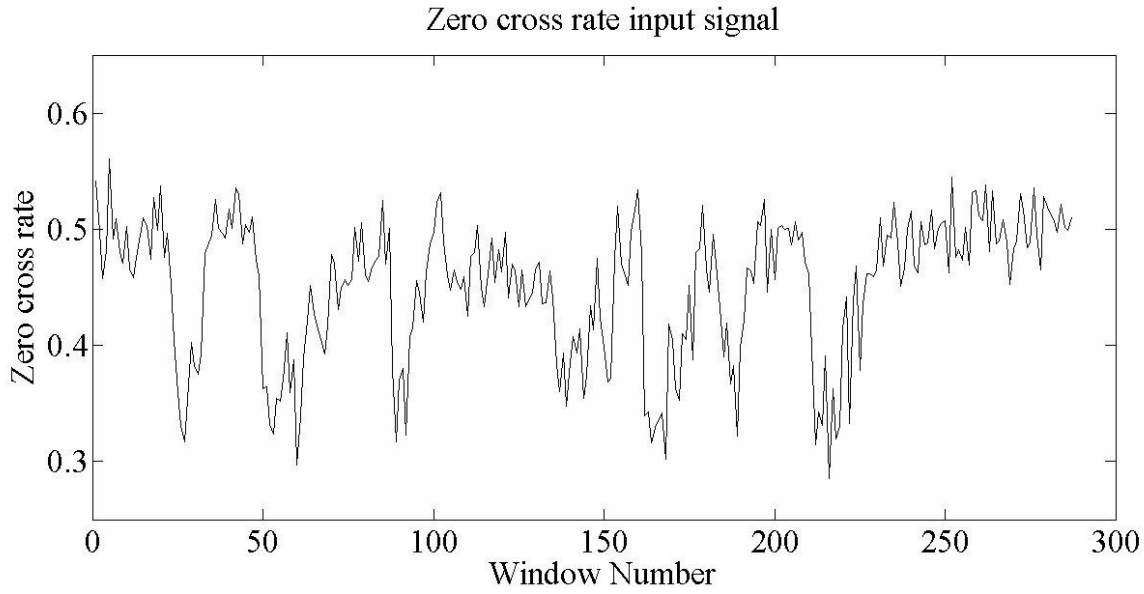


Fig 7.b

Fig 7.a shows the block diagram of a signal flow through algorithm. Fig 7.b shows the input signal(zero crossing rate of an input signal). Fig 7.c shows the output of a median smoothening filter in fixed point format and Fig. 7.d shows the output for fixed point format. The mean error observed was of order 10^{-8}. The Q5.26 format was used for simulation purposes.

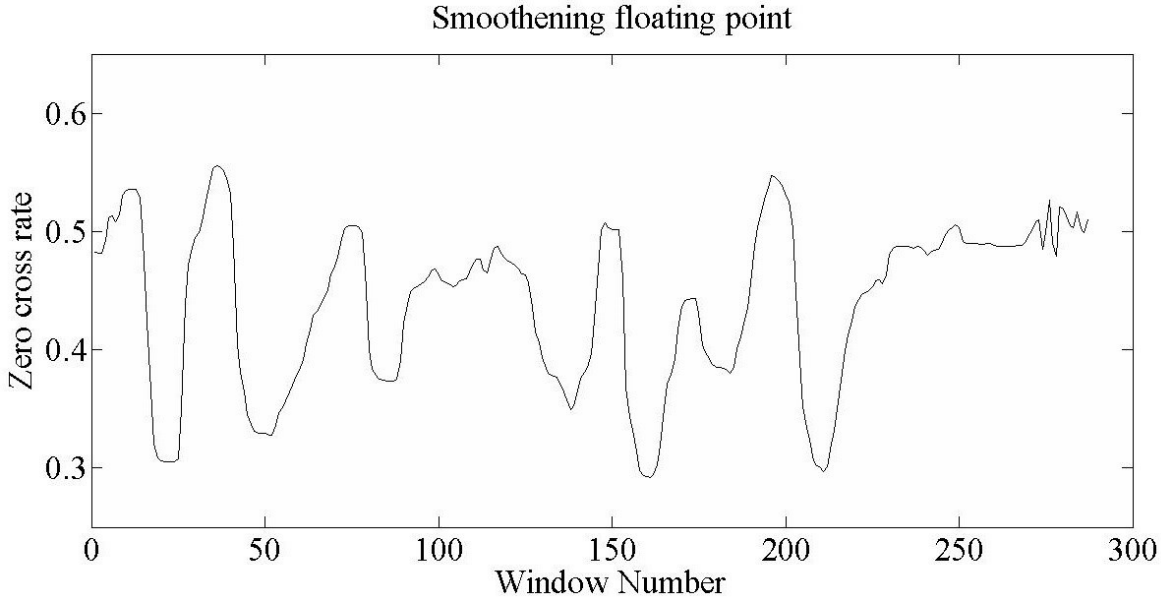


Fig 7.c

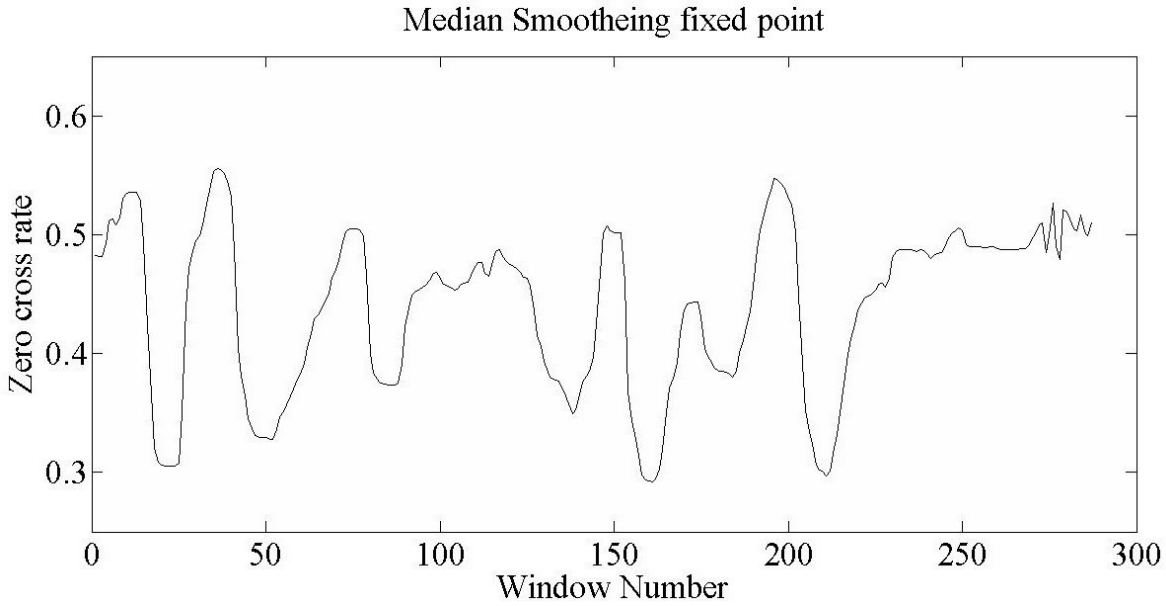


Fig 7.d

Algorithm 7:

Input: Speech signal, window size (in ms), Sampling Frequency
Output: Median smoothened signal
M = 10; *//Calculate the median of 10 samples*
Function Median_smothener(-------, -------)
{

a. Let N denote the number of samples in each window
N = 30 * Sampling_frequency / 1000
b. For i=5:1:N-5
{
For j=i-5:1:i+5
{

- Sort the samples.
- Pick the median sample and replace the i^{th} sample with the median value.

}
}
}

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0
Processed speech	<32,1,30>	1	-1	0

8) **Spectral subtraction for speech enhancement:** Effective performance of digital speech processors operating in practical environments may require suppression of noise from the digital waveform. Spectral subtraction offers a computationally efficient, processor independent approach to effective digital speech analysis. The method requires about same computation as high speed convolution, suppress stationary noise from speech by subtracting the spectral noise calculated during the non-speech activity. Since the algorithm resynthesizes the speech waveform, it can be used as a preprocessor to narrow-band voice communication systems, speaker recognition systems, or speaker authentication systems. The estimator estimates the amount of noise during non-speech activity and subtracts the estimate from the noisy speech spectrum.

Assumptions for noise:

- The background noise is acoustically added the speech.
- The background noise environment remains locally stationary to the degree that its spectral magnitude expected value just prior to the speech activity equals its expected value during speech activity.
- If the environment changes to a new state, there exist enough time (about 300ms) to estimate a new background noise spectral magnitude value before activity commences.
- The algorithm also requires a speech activity detector in case the noise is slowly varying.
- It is also assumed that a significant noise reduction can be achieved by just removing effect of noise from magnitude spectrum.

Additive noise model:

x(k) denotes a windowed noise speech signal.

s(k) denotes windowed speech signal

n(k) denotes windowed noise signal

x(k) = s(k) + n(k)

Take Fourier transform

$$X(e^{jw}) = S(e^{jw}) + N(e^{jw})$$

$$\mu(e^{jw}) = E\{|Ne^{jw}|\}$$

Now, the calculate mean can be utilized to reduce the noise content in spectral domain.

$$S(e^{jw}) = \ [|X(e^{jw})| - \ \mu(e^{jw})]\ e^{j\theta_x}$$

There are different ways to utilize the spectral subtraction algorithm to improve the speech content in input signal.

1) **Magnitude Averaging**: The spectral error equals the difference between the noise spectrum N and its mean μ, local averaging of the spectral magnitudes can be used to reduce the error. Hence replace $X|(e^{jw})|with\ \overline{|X(e^{jw})|}$ where

$$\overline{|X(e^{jw})|} = \frac{1}{M}\sum_{i=0}^{M-1} |X(e^{jw})|$$

Since speech is non-stationary only limited time averaging is allowed. Window size must be less than or equal to 30ms. Ideal size is between 25 to 30 ms. The major disadvantage of averaging is the risk of some temporal smearing of short transitory sounds.

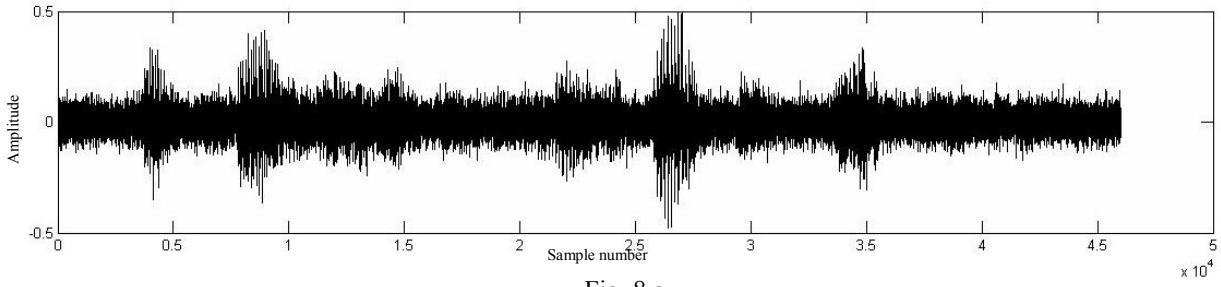


Fig. 8.a

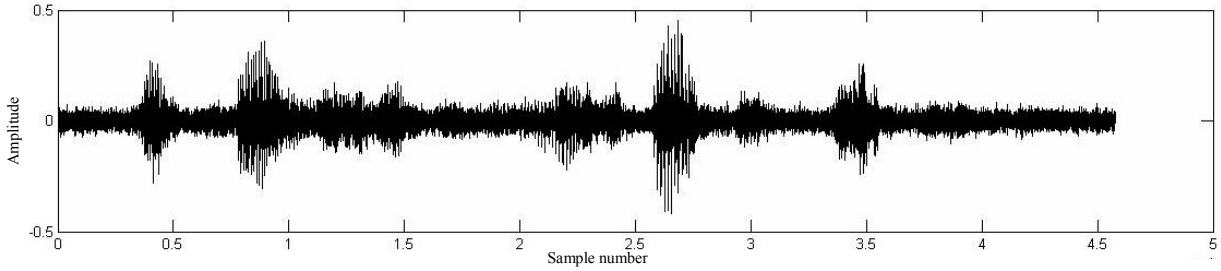


Fig. 8.b

In fig. 8.a the actual signal is plotted and in fig. 8.b the signal after spectral subtraction is being plotted.

Algorithm 8:

Input: Speech signal, Sampling Frequency
Output: Spectrally Enhanced Signal
Function Speech_enhancement (-------, -------)
{

a. Break the signal in windows of 25ms each with an overlap of 40 – 50 percent.
b. Let's suppose we have 100 ms of noise data. Break it in windows of 25 ms each with 40% overlap. Let's say we have 4 windows each of size N.

```
Val = 0;
For i=1:1:4
    {
      N_fft = fft(x, N);
      N_fft_abs = abs(N_fft);
      Sum (i) = 0;
            For j=1:1:N
                  {
                        Sum(i) = Sum(i) + N_fft_abs(j);
                        }
      Sum (i) = Sum (i) / N;
     Val = Val + Sum (i);
            }
   Val = Val/4;
```

c. No_of_windows = (size_of_speech)*2 / N -1;
// Add zeros to the end if required depending on the input speech signal

```
For i=1:1:No_of_windows
{
        X_fft = fft(x);
        X_fft_abs = abs (X_fft);
        X_fft_angle = angle(X_fft);
        For j = 1:1:N
                {
                        X_fft_abs = X_fft_abs – Val;
                        If(X_fft_abs < 0)
                                {
                                        X_fft_abs =0;
                                        }
                        }
```

// Remix the absolute calculated value with the angle. Let X_fft_new denote the combined signal now.

d. Take inverse fft.

e. Update the new signal by first 50% values from each recalculated signal and discard the last values.

Spectral subtraction performs well at all levels of SNR. The only issue is that if the noise signal is varying then value of Val has to recalculated after a certain time interval. We could use a separate mike to record the noise after certain predefined instants or use the speech activity detector provided in this reference to get a noise slot and use it to recalculate the value of Val.

9) **SVD(Singular Value Decomposition) based speech enhancement for white noise environment:** The SVD technique enhances a noisy signal by retaining a few of the singular values from the decomposition of an over-determined, over-extended data matrix. Here we shall be utilizing minimum variance technique for speech improvement. The algorithm works as:

- Let x = $\bar{x} + n$,where n is noise and x is a noisy speech signal.
- A Henkel matrix of order M*M is constructed of a windowed speech signal.
- The SVD is calculated for the Henkel matrix.
- Using voice activity detector, we find the variance of the noise from no speech frames.
- A F_{corr} matrix as given below is constructed. The F_{corr} matrix is a diagonal matrix with elements as given below

$$F_{corr} = diag(\left(1 - \frac{\sigma_{noise}^2}{\sigma_1}\right), \dots\dots\dots\dots\dots\dots, \left(1 - \frac{\sigma_{noise}^2}{\sigma_K}\right))$$

- The Henkel matrix is reconstructed with the relation

$$\bar{H} = U\, F_{corr} \sum_K V^T$$

- The sum of anti-diagonal elements is found and this gives us the updated speech.
- The size of window should not be more than 25 to 30ms as speech signal acts as stationary signal in that interval.

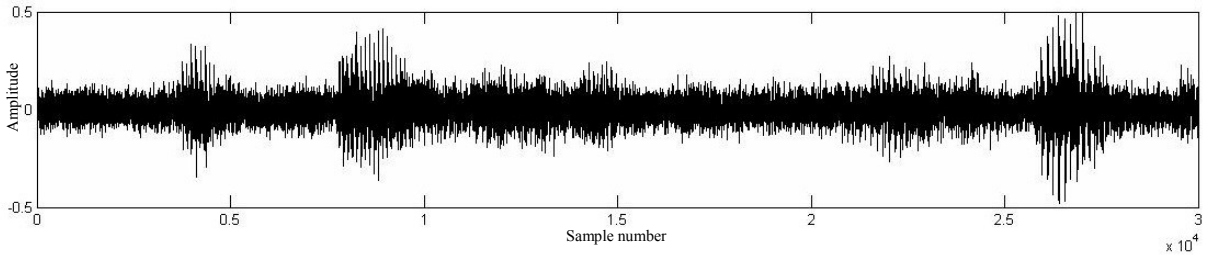


Fig 9.a

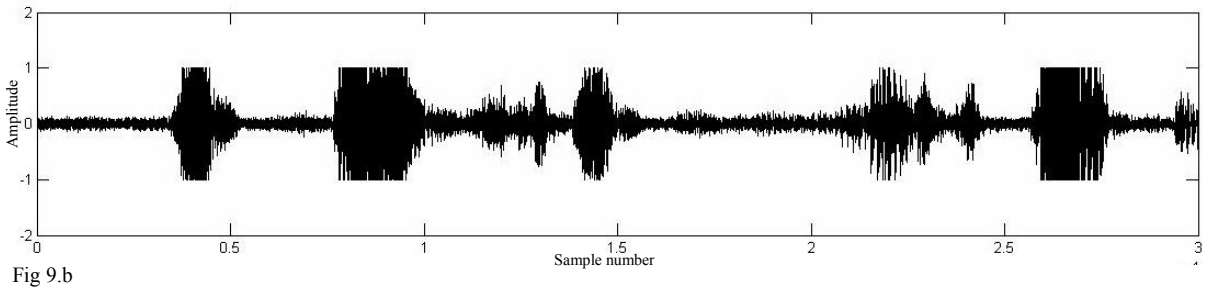


Fig 9.b

In fig 9.a shows the actual signal and fig. 9.b shows the processed signal by SVD method.

Algorithm 9:

Input: Speech signal, Sampling Frequency
Output: SVDEnhanced Signal
Function Speech_enhancement (-------, -------)
{

a. Break the signal in windows of 25ms each with an overlap of 40 – 50 percent.
b. Let's suppose we have 100 ms of noise data. Break it in windows of 25 ms each with 40% overlap. Let's say we have 4 windows each of size N.

```
Val = 0;
        For i=1:1:4
            {
              Var(i) = 0;
                  For j=1:1:N
                        {
                              Var(i) = Var(i) + x(j)*x(j);
                              }
                Val = Val + Var (i);
                      }
            Val = Val/4;
```

c. No_of_windows = ((size_of_speech)*2 / N) -1;

// Add zeros to the end if required depending on the input speech signal

For i=1:1:No_of_windows

{

- Form a Henkel matrix as defined above o f a windowed signal.
- Calculate SVD of the matrix.
- Calculate the Fcorr matrix as defined below
- $F_{corr} = diag(\left(1 - \frac{\sigma_{noise}^2}{\sigma_1}\right), \ldots\ldots\ldots\ldots\ldots\ldots, \left(1 - \frac{\sigma_{noise}^2}{\sigma_K}\right))$
- Recalculate the henkel matrix using

$$\bar{H} = U\, F_{corr} \sum_K V^T$$

// Now we need to recalculate the updated signal values by taking the sum of anti- diagonal elements.

```
For i=1:1:N/2
    {
     k =  i;
    x(i) = 0;
    For j=1:1:i
       {
         x(i) = x(i) + h(k,j);
         k = k-1;
             }
      x(i) = x(i) / i;
    }
}
```

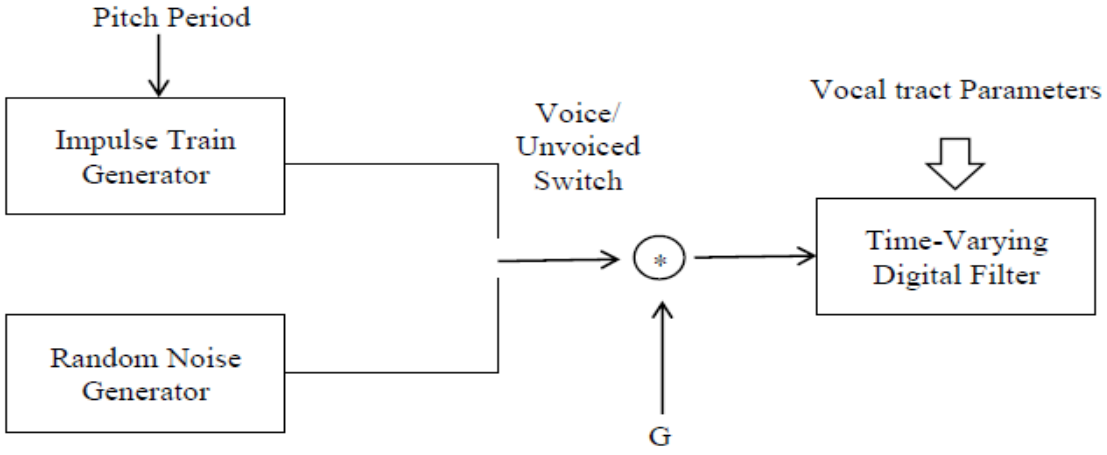
// The SVD method of speech enhancement is applicable only upto 15db. It does result in some distortion in the enhanced speech, but is a better way to enhance speech than spectral subtraction.

10) **Linear predictive coding (LPC) of speech:** Linear predictive analysis is predominant technique used these days to estimate the basic speech parameters, e.g. pitch, formant, spectra, vocal tract functions, and for representing speech for low bit rate transmission and storage. The beauty of the algorithm is reduced computation and high accuracy. The basic idea behind LPC is that a speech sample can be approximated as a linear combination of past speech samples. By minimizing the sum of the squared differences between the actual speech samples and the linearly predicted ones, a unique set of predictor coefficients can be determined. The various algorithms that are used for LPC calculations are

a. The covariance method
b. The autocorrelation method
c. The inner product method

Basic principle of LPC: The composite effect of radiation, vocal tract, and glottal excitation are represented by a time varying digital filter whose steady state function s of the form

$$H(z) = \frac{S(z)}{U(z)} = \frac{G}{1-\sum_{k=1}^{p} a_k z^{-1}}$$



Block Diagram of simplified model for speech production.

The basic problem of linear prediction analysis is to determine the set of predictor coefficients directly from speech signal in such a manner as to obtain a good estimate of the spectral properties of the speech signal. Because of time varying nature of speech signal, the coefficients must be estimated over short segments of the speech.

1) **The Autocorrelation Method:** A window of length N is assumed and $s_n(m)$ is assumed zero outside the interval$0 \leq m \leq N-1$. This can be expressed as

$$s_n(m) = s_n(m+n)w(m)$$

where w(m) is a window function. The prediction error can be large at start of the window because we are trying to predict the signal from samples that have arbitrarily been set to zero. Similarly, the error can be large at the end of the interval because we are trying to predict zero from the samples that are non-zeros. For this reason, usually a window which tapers at the ends is used.

$$\emptyset(i,k) = \sum_{m=0}^{N+p-1} s_n(m-i)s_n(m-k)$$

It can be seen that $\emptyset(i,k)$ is identical to the short time autocorrelation function.

$$R_n(k) = \sum_{m=0}^{N-1-k} s_n(m)s_n(m+k) \quad 1 \leq i \leq p; 0 \leq k \leq p$$

The minimum mean square error prediction leads us to a matrix form as expressed below

$$\begin{bmatrix} R_n(0) & R_n(1) & R_n(2) & \dots & R_n(p-1) \\ R_n(1) & R_n(0) & R_n(1) & \dots & R_n(p-2) \\ \dots & \dots & \dots & \dots & \dots \\ R_n(p-1) & R_n(p-2) & R_n(p-3) & \dots & R_n(0) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \dots \\ R_n(p) \end{bmatrix}$$

Here p signifies number of coefficients to be calculated. The above matrix can be solved by Durbin's recursive solution.

2) **The Covariance Method:** A window of length N is assumed and $s_n(m)$ is assumed zero outside the interval $0 \leq m \leq N-1$. This can be expressed as

$$s_n(m) = s_n(m+n)w(m)$$

where w(m) is a window function.

$$\varphi_n(i,k) = \sum_{m=0}^{N-1} s_n(m-i)s_n(m-k)$$

By changing the index of summation we can express $\varphi_n(i,k)$

$$\varphi_n(i,k) = \sum_{m=0}^{N-1-k} s_n(m)s_n(m+k-1) \;\; 1 \leq i \leq p; 0 \leq k \leq p$$

The above equation call for value of $s_n(m)$ outside the interval $0 \leq m \leq N-1$. Hence to reduce the mean square error we have to supply those values. The final matrix is arranged as

$$\begin{bmatrix} \varphi_n(1,1) & \varphi_n(1,2) & \varphi_n(1,3) & \dots & \varphi_n(1,p) \\ \varphi_n(2,1) & \varphi_n(2,2) & \varphi_n(2,3) & \dots & \varphi_n(2,p) \\ \dots & \dots & \dots & \dots & \dots \\ \varphi_n(p,1) & \varphi_n(p,2) & \varphi_n(p,3) & \dots & \varphi_n(p,p) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} \varphi_n(1,0) \\ \varphi_n(2,0) \\ \dots \\ \varphi_n(p,0) \end{bmatrix}$$

The above matrix equations can be solved by choleskys decomposition algorithm.

3) **Matrix method:** The most simple method is to arrange the $s_n(m)$ in the form of matrix as :

$$A= \begin{bmatrix} s_n(0) & s_n(1) & s_n(2) & \dots & s_n(p-1) \\ s_n(1) & s_n(2) & s_n(3) & \dots & s_n(p) \\ \dots & \dots & \dots & \dots & \dots \\ s_n(N-p-1) & s_n(N-p) & s_n(N-p+1) & \dots & s_n(N-1) \end{bmatrix}$$

$$B = \begin{bmatrix} s_n(p) \\ s_n(p+1) \\ \dots \\ s_n(N-1) \end{bmatrix}$$

Let C be a matrix that contains the LPC coefficients

A*C = B

Multiply A' on both sides

A'*A*C = A'*B

Taking $(A' * A)^{-1}$

C = $(A' * A)^{-1}$*A'*B;

The above method though simple is computationally very intensive, when matrix order is larger. Hence the other two methods are preferred over this one.

LPC coefficients	Autocorrelation Method	Covariance Method	Matrix Method
α_1	2.0167	2.0133	2.0199
α_2	-2.0074	-1.9847	-1.9986
α_3	1.1909	1.1145	1.1289
α_4	-0.2939	-0.2081	-0.2009
α_5	0.3446	0.3126	0.2953
α_6	-0.8535	-0.8859	-0.8859
α_7	1.0731	1.1269	1.1498
α_8	-0.5127	-0.5218	-0.5496
α_9	-0.0334	-0.0500	-0.0572
α_{10}	0.0173	0.0155	0.0562
α_{11}	0.0287	0.0579	0.0158
α_{12}	-0.0921	-0.1177	-0.0985

LPC coefficients	Autocorrelation Method (fixed point)	Covariance Method(fixed point)	Matrix Method(fixed point)
α_1	2.0167	2.0133	2.002
α_2	-2.0092	-1.9847	-1.9760
α_3	1.1931	1.1145	1.1329
α_4	-0.2972	-0.2082	-0.1999
α_5	0.3491	0.3127	0.2921
α_6	-0.8588	-0.8860	-0.9201
α_7	1.0786	1.1270	1.1106
α_8	-0.5183	-0.5219	-0.5397
α_9	-0.0278	-0.0499	-0.0643
α_{10}	0.0120	0.0154	0.0413
α_{11}	0.0329	0.0580	0.0158
α_{12}	-0.0936	-0.1177	-0.0769

Q5.26 format was used for fixed point simulations. As is visible from the table that a minute variations do occur in LPC calculations and covariance method provide most accurate results in fixed point simulations and is also least time consuming. The Matrix method provides least accurate results in fixed point due to matrix inversion. Hence it is least preferred one , though it is simple to understand and implement.

Algorithm 10:

Input: Speech signal, Sampling Frequency
Output: LPC coefficients
Function LPC_calculator (-------, -------)
{

a. Let N denote the number of samples in each window
   N = 30 * Sampling_frequency / 1000
b. Find the autocorrelation sequence as specified in autocorrelation method of LPC calculation.
c. Construct the matrix and use Durbin's recursive algorithm to get LPC coefficients.

OR

b. Find the Covariance Matrix as specified in covariance method of LPC calculation.
c. Construct the Covariance matrix and use choleskys decomposition algorithm to get LPC coefficients.

OR

b. Construct the inner product matrix as shown in matrix method.
c. Multiply both sides by A transpose
d. Take inverse to calculate the LPC coefficients.

// All three methods should provide ideally same results, and they do but within a certain error of order of 10^{-4}*. Matrix method requires lot of memory and inversion of a matrix is not that simple when all values are very small, especially when using fixed point arithmetic. Hence matrix method should be avoided as much as possible. Lpc coefficients are used in number of places in speech processing algorithms like pitch estimation, speech compression, Formant frequency estimation, speaker identification.*

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0
Auto correlation	<32,5,26>	8.9264	0	4.5623
K	<32,5,26>	8.9264	-10.1060	-1.0314
alpha	<32,5,26>	2.6765	-3.0421	-0.8734
E	<32,5,26>	1.4322	2.3859e-06	0.6583

Autocorrelation LPC method

Algorithm 10.1:

Durbin's Recursive Solution

The equations we got were

$$\sum_{k=1}^{p} \alpha_k R_n(|i-k|) = R_n(i) \qquad 1 \leq i \leq p$$

Let

$E^0 = R(0);$

For i=1:1:p

{

$k_i = \{ R(i) - \sum_{j=1}^{i-1} R(i-j)\}/E^{i-1};$

$\alpha_i^{(i)} = k_i;$

$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{(i-1)};$

$E^{(i)} = (1-k_i^2)E^{(i-1)};$

}

For i=1:1:p

{

$\alpha_j = \alpha_j^{(p)};$

}

// The α_j denotes the LPC coefficients values. For choleskys decomposition solution please refer to any standard book on numerical methods.

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0
Phi	<32,5,26>	11.5726	-9.0429	4.5623
A	<32,5,26>	11.5726	-8.3199	1.3452
V	<32,5,26>	1	0	0.8734
D	<32,5,26>	11.0759	2.3865e-06	6.7345
Yb	<32,5,26>	1.1765	-2.9164	-0.8548
alpha	<32,5,26>	10.3520	-13.6269	-0.2386

LPC Covariance Method

11) **Short time average Magnitude difference equation (AMDF):** The computation of autocorrelation function involves considerable arithmetic. AMDF is a technique that eliminates the need for multiplications

$$d(n) = x(n) - x(n-k)$$

For a truly periodic function with period P, the value of d(n) would be zero for . For short segments of voiced speech it is reasonable to expect d(n) to be small whenever n is closer to P. This is exploited to determine the pitch period of the signal. Since AMDF is implemented using addition, subtraction and absolute value operation, hence for fixed point calculations it can turn out be quite useful. For above mentioned reasons AMDF is used in numerous multiple real time speech processing algorithms.

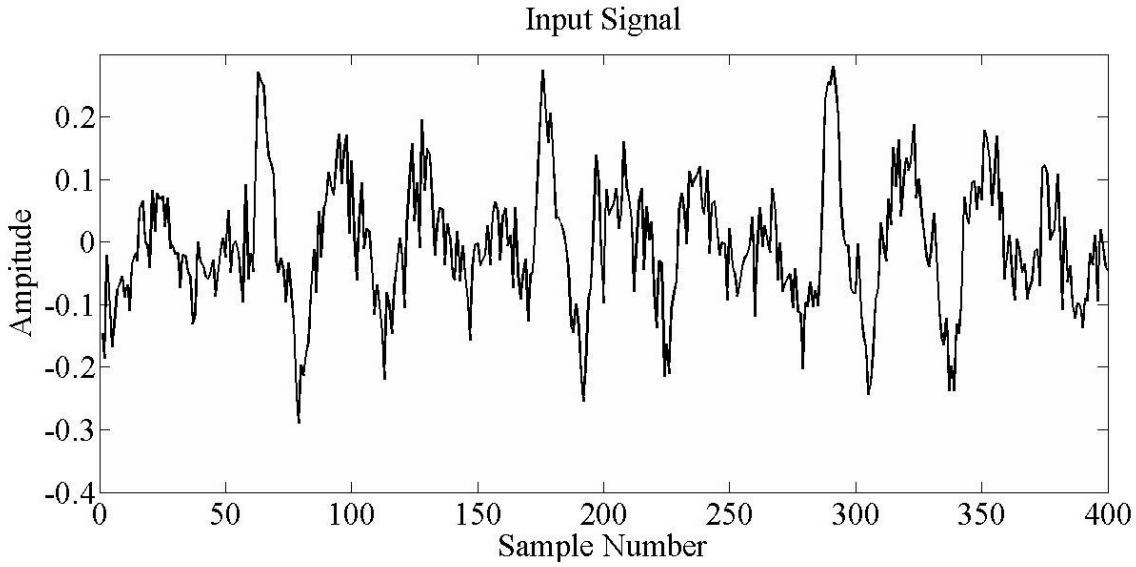


Fig. 11.a

Fig. 11.a shows a window of speech signal, Fig. 11.b shows the AMDF function for a floating point format and Fig. 11.C shows the output of AMDF function in fixed point format.

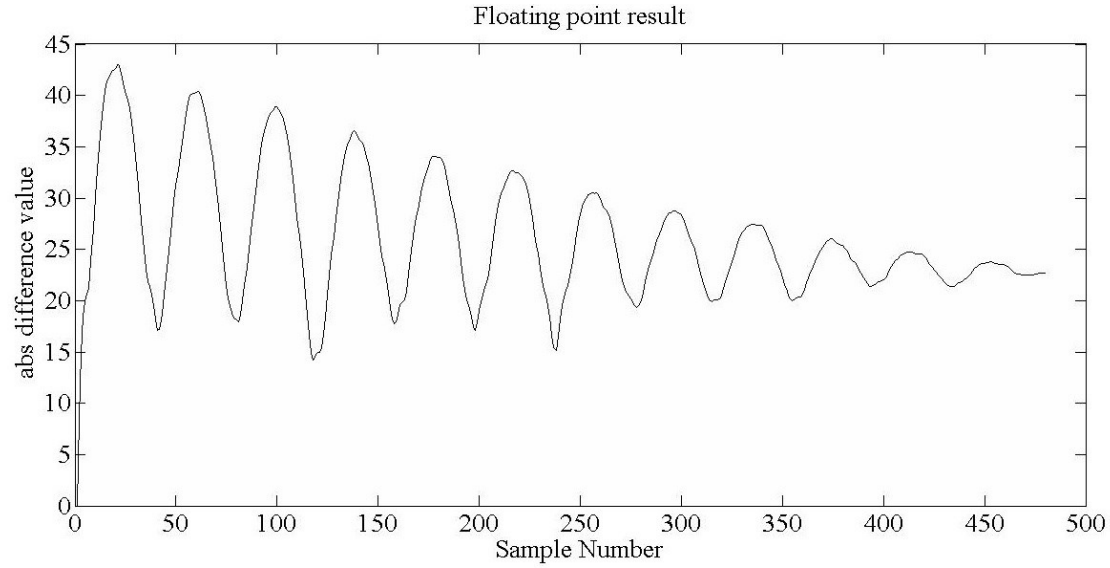


Fig. 11.b

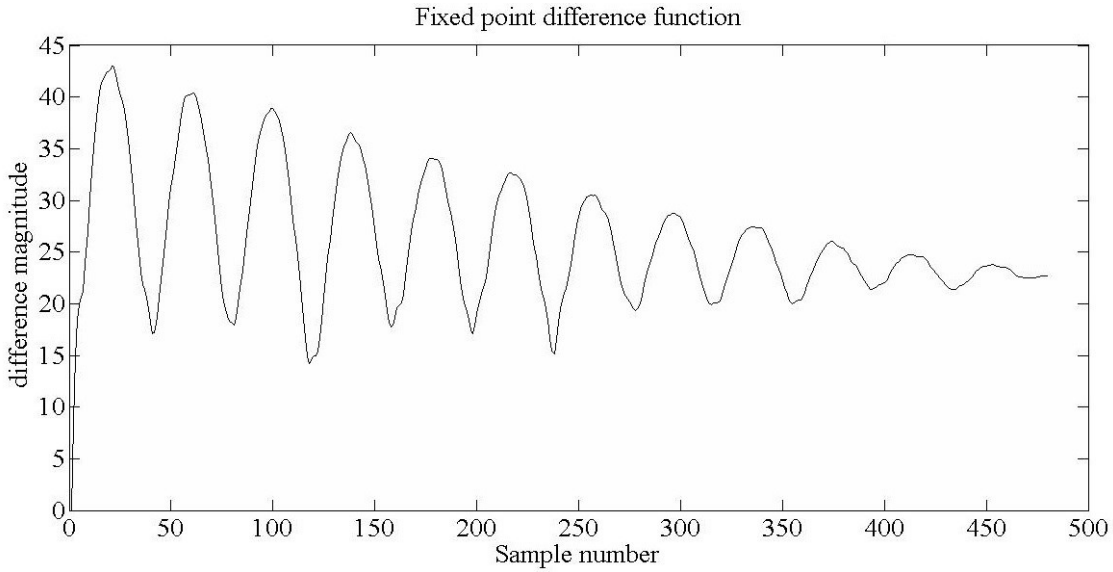


Fig. 11.c

Variable	Word-length	Range		
		Max	Min	Mean
Speech Signal	<32,1,30>	1	-1	0
Difference	<32,8,23>	105.8868	0	15.0653

Algorithm 11:

Input: Speech signal, Sampling Frequency
Output: STAMD aaray
Function STAMD_function(-------, -------)
{

a. Let N denote the number of samples in each window

N = 30 * Sampling_frequency / 1000

```
For i=1:1:N
    {
      diff(i) = 0;'
      For j=1:1:N
            {
                if ( (j – i) <0)
                    diff(i) = diff(i) + abs(x(j));
                else
                    diff(i) = diff(i) + abs( x(j) – x(j-i));
                    }
            }
    Return ( diff array);
    }
```

// Implementation of AMDF function in fixed point is a big issue as the numbers being computed are too small and truncation results in huge erorrs. Hence its better to use floating point format for AMDF function or use 64 bit accumulator function for calculation diff sum and then truncate at the end of all calculations.

12) **FORMANT FREQUENCY CALCULATIONS:** Formants are defined by Gunnar Fant as "the spectral peaks of the sound spectrum |P(f)|". In acoustics generally, a very similar definition is widely used: the Acoustical Society of America defines a formant as: "a range of frequencies [of a complex sound] in which there is an absolute or relative maximum in the sound spectrum. In speech science and phonetics, however, a formant is also sometimes used to mean an acoustic resonance of the human vocal tract. Thus, in phonetics, formant can mean either a resonance or the spectral maximum that the resonance produces. Formants are often measured as amplitude peaks in the frequency spectrum of the sound, using a spectrogram or a spectrum analyzer and, in the case of the voice; this gives an estimate of the vocal tract resonances. In vowels spoken with a high fundamental frequency, as in a female or child voice, however, the frequency of the resonance may lie between the widely-spaced harmonics and hence no corresponding peak is visible.

Today, virtually all high-performance speech recognition systems are based on some kind of mel-cepstral coefficients or filter bank analysis. So, we do not expect that in the near future formant-based parameters will be competitive. Nevertheless, there might be specific aspects due to which formant-based parameters are attractive, as listed below.

- Formants are considered to be robust against channel distortions and noise.
- Formant parameters might provide a means to tackle the problem of a mismatch between training and testing conditions.
- There is a close relation of formant parameters to model based approaches to speech perception and production.

Fig. 12.a shows the window of an input signal on which algorithm was applied. Fig 12.b shows the magnitude response of fast Fourier transform performed on the signal and Fig 12.c shows the angle of fast Fourier transform. Fig 12.d shows the log of magnitude of fast Fourier transform and Fig 12.e shows the log of magnitude of FFT of whole signal.

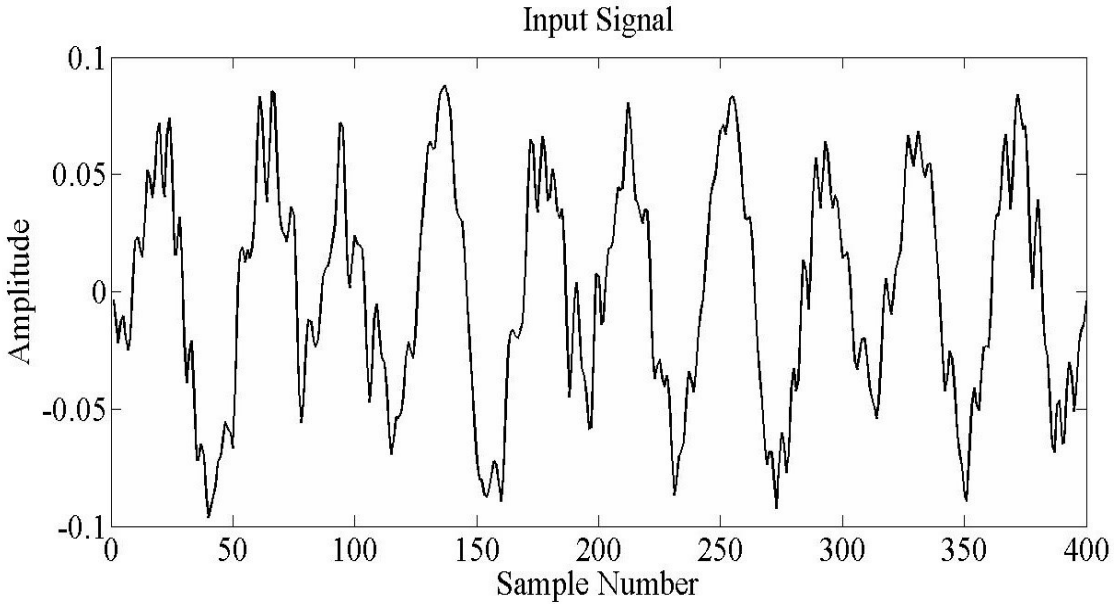


Fig. 12.a

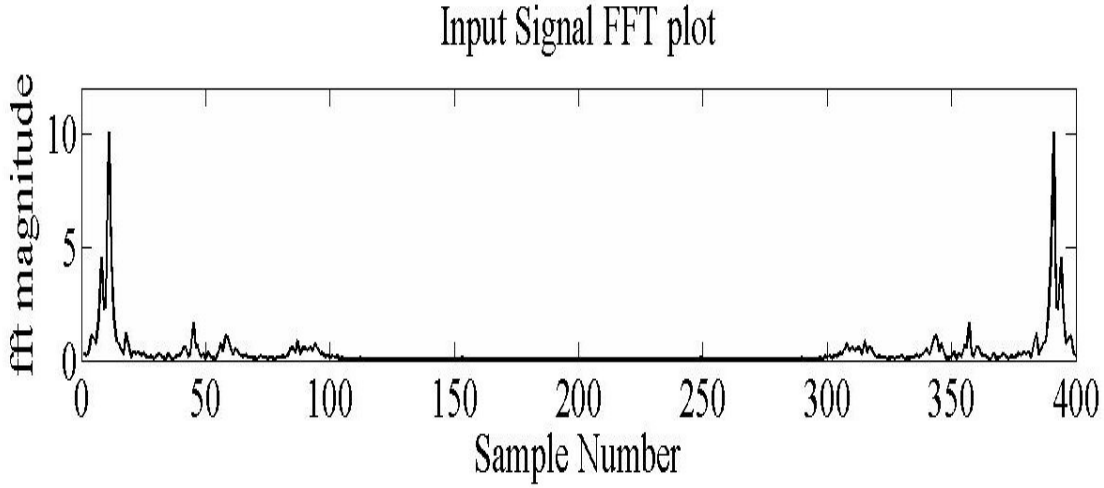


Fig. 12.b

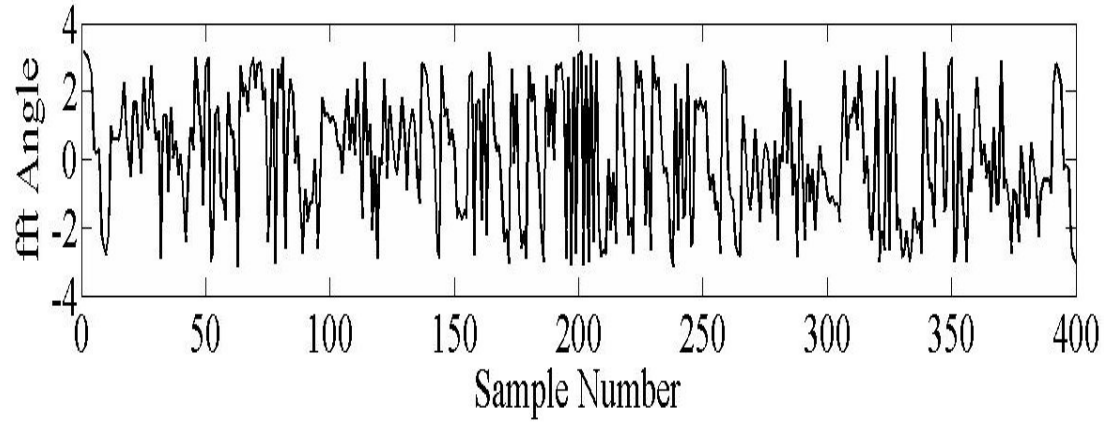


Fig. 12.c

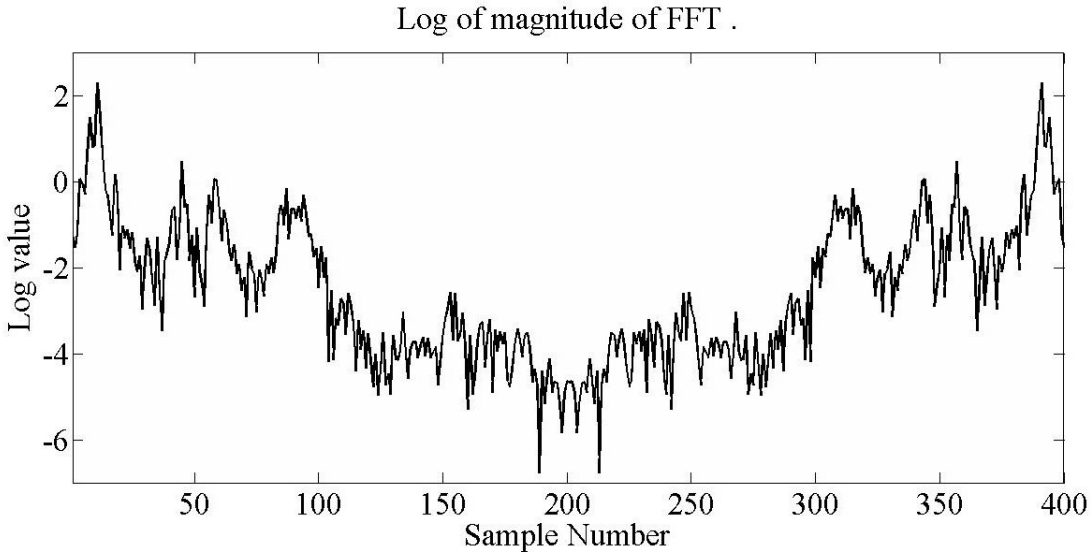


Fig. 12.d

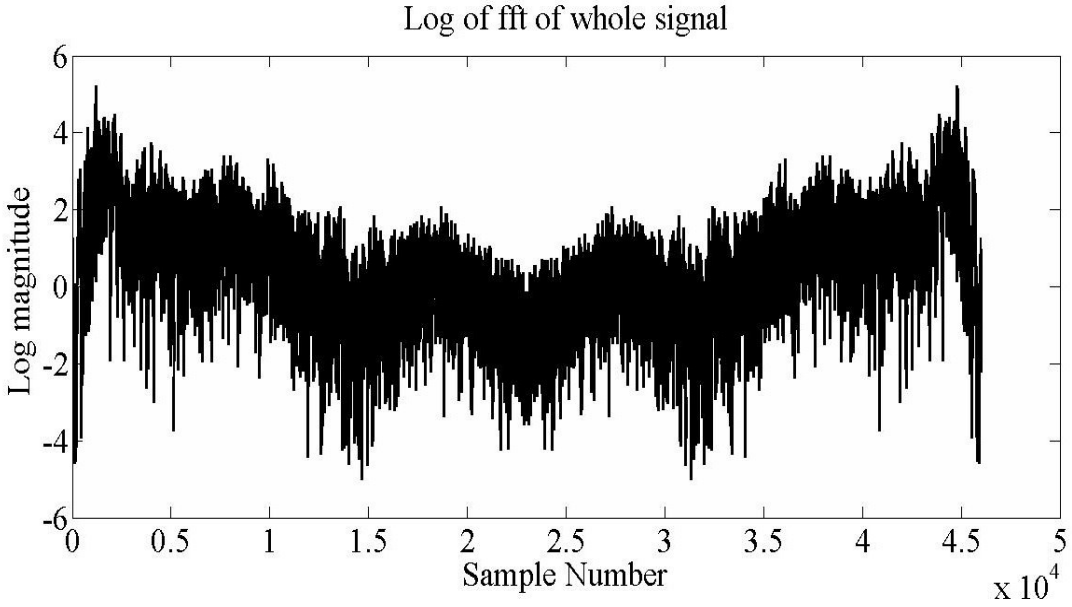


Fig. 12.e

Graphs Fig 12.d and 12.e are almost identical with small variations. The reason being that both depicts the cestrum nature of the speech which is almost constant for a person with minute variations as the shape of vocal tract varies very little in two consecutive windows due to muscle and skin adjustment.

Algorithm 12:

Input: Speech signal, Sampling Frequency
Output: Formant frequencies
Function Formant_frequency_function(-------, -------)
{

a. Break the signal into no of windows, with or without overlap depending on the application.
b. Let's say we have a window of size N (must be more that at least 25ms of speech data).
c. Use either of the algorithms mentioned in LPC method (Algorithm 10) to find the LPC parameters of the window.
d. Construct the predictor polynomial as mentioned below

$$A(z) = 1 - \sum_{k=1}^{p} \alpha_k Z^{-k}$$

Where p is no of LPC coefficients (normally p=10 or 12 works fine for first three formant frequencies).

e. Find the roots of the predictor polynomial (complex conjugate roots wil occur if p= even).
f. Retain the roots with positive imaginary part. Hence now number of roots will be halved as other half are discarded.
g. Determine the angle corresponding to those roots, given by formula:

$$Angle(i) = Tan^{-1}\left(\frac{imag(root)}{real(root)}\right);$$

h. Convert this angular frequency from radians to Hz using the formula:

$$Frequency(i) = angle(i) * \frac{Fs}{2 * \pi};$$

i. Get the corresponding bandwidth of the Formant frequencies using

$$Bandwidth(i) = \frac{-1}{2} * \frac{Fs}{(2 * \pi)} * \log\left(abs\left(root(i)\right)\right);$$

// Use the criteria that formant frequencies should be greater than 90 Hz with bandwidths less than 400 Hz to determine the formants.

```
i = 1;
for k = 1:1:No_of_roots
        {
          if (Frequencies(k) > 90 && Bandwidth(k) <400)
              {
                    Formant(i) = Frequecny(k);
                    Bandwidth(i) = Bandwidth(k);
                    i = i+1;
                    }
              }
}
```

Algorithm 12.1:

Input: Polynomial in form

$f(x) == a_1 + a_2x + a_3x^2 + \cdots \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + a_{n+1}x^n$

Output: All roots of the polynomial

Function Polynomial_Roots(……, ……)

{

- Pick $\in > 0$, m > 0. Set h(x) = f(x) and k=1;
- Do

{

a. Using $x_0 = 0$, apply algorithm12.2 to h(x) to find a root y.
b. Using $x_0 = y$, apply algorithm 12.2 to f(x) to find a root x_k.
c. If x_k is real then

{

1. Apply algorithm 12.3 to divide h(x) by $(x - x_k)$.
2. Set h(x) = g(x) where g(x) is the quotient polynomial.
3. Set k=k+1;

}

Else

{

1. Set $x_{k+1} = x_k^*$.
2. Apply algorithm 12.4 to divide h(x) by $(x - x_k)(x - x_{k+1})$
3. Set h(x) = g(x) where g(x) is the quotient polynomial.
4. Set k=k+2;

}

}

while $(k \leq n)$

}

// The basic approach is to find one root at a time, deflating the polynomial f after each root is found. Subsequent roots are found using the deflated polynomial. If the real root is found, the polynomial is deflated by a linear factor using linear synthetic division. Otherwise, it is deflated by a linear factor using quadratic synthetic division. Ultimately, the deflated polynomial is of degree one or two at which point closed form expressions for the roots can be used. The deflation process works best when the roots with smaller magnitude are removed first. Consequently $x_0 = 0$ *is used as an initial guess when Lagurre's method is applied.*

In order to increase the numerical accuracy of the estimated roots, a two-step process is used. First a root is found by applying the Laugerre's method to the deflated polynomial. This root is then used as initial guess and laguerre's method is applied a second time to the original polynomial f. This latter step is called polishing of the root.

Algorithm 12.2:

Input: Polynomial in form

$f(x) == a_1 + a_2x + a_3x^2 + \cdots \ldots \ldots \ldots \ldots \ldots \ldots \ldots + a_{n+1}x^n$

Output: A real root or a complex root of the polynomial

Function Laguerre's_method(……, ……)

{

- Set $\in > 0$, m > 0, x_0. Set k = 1. Compute $f_o = f(x_0)$.
- If n = 1, set $x_1 = -\frac{a_1}{a_2}$ and return.
- Do

{

1. Compute $f_1 = f'(x_0)$.
2. Compute $f_1 = f''(x_0)$.
3. Compute $g = f_1^2 - nf_0f_2/(n-1)$
4. Compute $d_1 = f_1 + (n-1)\sqrt{g}$
5. Compute $d_1 = f_1 - (n-1)\sqrt{g}$

If ($|d_1| \geq |d_2|$)

{

$x_1 = x_0 - nf_0/d_1$

}

Else

{

$x_1 = x_0 - nf_0/d_2$

}

Compute

{

$x_0 = x_1$

$f_0 = f(x_0)$

k = k + 1;

}

}

While ($|f_0| \geq \in$) and $k \leq m$)

Set $x = x_1$

}

//The laguerre's method uses both the first and second derivatives of f. This result in a very fast algorithm that exhibits cubic convergence to simple roots that may be real or complex. For polynomials with only real roots, Laguerre's method is guaranteed to converge to a root from any starting point x_0.

The above algorithm terminates when the prescribed error tolerance on |f(x)| has been achieved or a maximum number of iterations m has been profounded.

Algorithm 12.3:

Linear Synthetic Division

Input: Polynomial in form of an array containing factors $f(x) = a_1 + a_2x + a_3x^2 + \cdots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots + a_{n+1}x^n$ & a root c.

Output: Polynomial inform of $f(x) = b_1 + b_2x + b_3x^2 + \cdots \ldots \ldots \ldots + b_nx^{n-1}$ and remainder d

Function Linear_synthetic_division(-------, -------)

{

- Set $b_n = a_{n+1}$
- For k = 1 to n-1 compute

  {

  $b_{n-k} = a_{n-k+1} + cb_{n-k+1};$

  }
- Set $d = a_1 + cb_1;$

  }

If d = 0 , it implies perfect division and hence root c is a root of given polynomial. This algorithm is used when we get a real root of a polynomial which has real coefficients.

Algorithm 12.4:

Quadratic Synthetic Division

Input: Polynomial in form of an array containing factors $f(x) = a_1 + a_2x + a_3x^2 + \cdots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots + a_{n+1}x^n$ & two complex roots

Output: Polynomial inform of $f(x) = b_1 + b_2x + b_3x^2 + \cdots \ldots \ldots \ldots + b_nx^{n-1}$ and remainder as rx+s

Function Quadratic_synthetic_division(-------, -------)

{

- p = 2 * Real(root_1);
- -q = root_1 * root_2
- Set $b_{n-1} = a_{n+1}$ and $b_{n-2} = a_n + pb_{n-1};$
- For k = 3 to n-1 compute

  {

  $b_{n-k} = a_{n-k+2} + pb_{n-k+1} + qb_{n-k+2};$

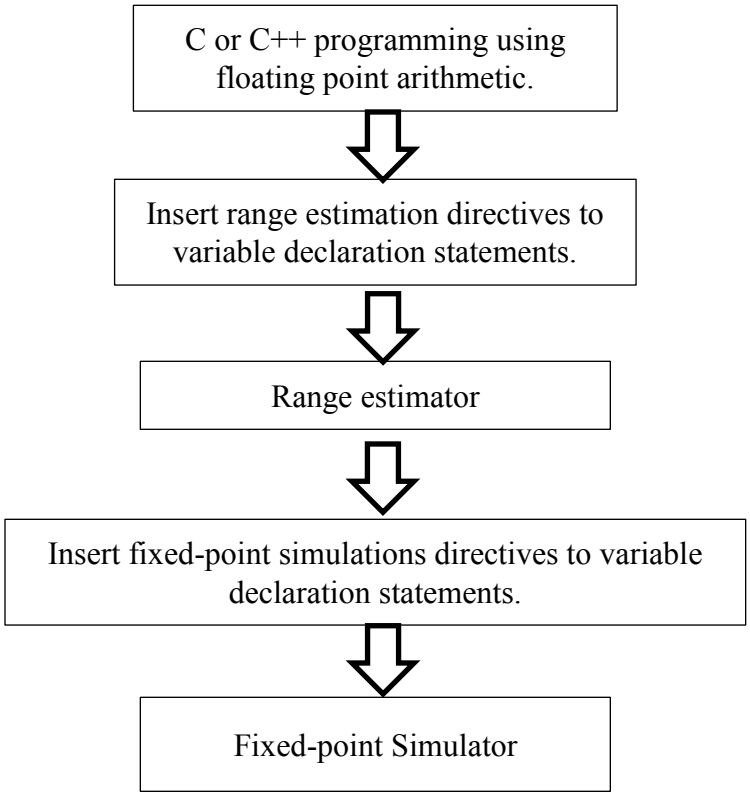
  }
- Set $r = a_2 + pb_1 + qb_2$ and $s = a_1 + qb_1;$

  }

If both r = 0 and s=0, it implies perfect division and hence roots root_1 and root_2 c are roots of given polynomial. This algorithm is used when we get a two conjugate complex roots of a polynomial which has real coefficients.

Chapter: 3

Fixed- Point optimization utility for C and C++: While most digital signal processing algorithms are developed using floating-point arithmetic, their implementation using very large scale implementation (VLSI) or fixed-point digital signal processors usually requires fixed point arithmetic for the sake of hardware cost and speed. However, fixed-point implementation can suffer from excessive finite word length effects, due to overflows and quantization noise, unless all signals are scaled properly and enough word lengths are assigned. This utility consists of the range estimator and the fixed point simulator. The proposed procedure for developing fixed point algorithms is shown in Fig. 13.1. Users develop C or C++ models with floating-point arithmetic and mark the variables whose fixed-point behaviour is to be examined with the range estimation directives. The range estimator then finds the statistics of internal signals throughout the floating-point simulation using real inputs and determines scaling parameters.



Matlab Functions references: The follow up text describes the way to call the Matlab functions as provided in the CD attached with the reference:

Common terminologies:

- Input: Array containing input speech signal.
- Fs: Frequency of input signal.
- wind_size: window size for processing in milliseconds.
- Overlap: Defines the amount of overlap in between two successive windows.

1) Short time energy function:

Average_energy = ST_energy (input_speech, fs, wind_size)

Average_energy = ST_energy_fix (input_speech, fs, wind_size)

The output will be an array containing the average energy of each window, with the index of array refereeing to the window number.

2) Short time average magnitude function:

Average_magnitude = ST_abs_magnitude(input, fs, wind_size)

Average_magnitude = ST_abs_magnitude(input, fs, wind_size)

The output will be an array containing the average magnitude of each window, with the index of array refereeing to the window number.

3) Short time zero cross rate function:

Zero_cross_Rate_out = ST_zero_cross_rate(input, fs, wind_size)

Zero_cross_Rate_out = ST_zero_cross_rate(input, fs, wind_size)

The output will be an array containing the average zero cross rate of each window, with the index of array refereeing to the window number.

4) Short time auto correlation function:

autocorrelation_out = ST_autocorrelation(input, fs, wind_size)

autocorrelation_out = ST_autocorrelation(input, fs, wind_size)

The output will be an 2D array containing the autocorrelation values of each window.

5) Speech activity Detector:

[start_window,end_window] = Speech_activity_detector(input, fs, wind_size)
[start_window,end_window] = Speech_activity_detector_fix(input, fs, wind_size)

Start_window defines the window number which contains speech and End_window defines the end of the corresponding speech. The speech lies in between Start_window(i) and End_window(i).

6) Pitch estimation using autocorrelation: Available in fixed point

Pitch_out = Pitch_estimator(input,fs,wind_size, overlap)

All calculations are being done in fixed integer form , so the code for floating point and fixed point is same. The output is Pitch_out is an array which defines the pitch of each window

NOTE: Please pass the speech signal through a low pass filter with cut off of 900Hz for best results.

7) Median Smoothening:

Data_output = Median_smooth(input,fs,N0)

Data_output = Median_smooth_fix(input,fs,N0)

N0 defines number of samples over which media is calculated. Should be an odd number.

Data_out is an array which has been smoothened using media filtering.

8) **LPC Calculation:** Available in two forms

- ***LPC_output = LPC_autocorr(input,fs,wind_size,overlap,N0)***
- ***LPC_output = LPC_autocorr_fix(input,fs,wind_size,overlap,N0)***
- ***LPC_output = LPC_covariance(input,fs,wind_size,overlap,N0)***
- ***LPC_output = LPC_covariance(input,fs,wind_size,overlap,N0)***

  N0 defines number of LPC parameters to be calculated

LPC_output is a 2D array which contains N0 LPC parameter values for each window.

9) **Short time average magnitude difference function:**

Ouput_data = ST_avg_mag_diff(input, fs, wind_size, overlap)

Output_data = ST_avg_mag_diff(input, fs, wind_size, overlap)

The output will be an 2D array containing the difference values of each window for varying values of sample difference.

10) **Formant Frequency Calculation:**

- *Formant_output = formant_freq_autocorr(input,fs)*
- *Formant_output = formant_freq_covariance(input, fs)*

The output will be an 2D array containing the formant frequencies for each window.

C++ Functions references: The follow up text describes the way to call the C++ functions as provided in the CD attached with the report:

Common terminologies:

- Input: Provide the path of text file containing speech data in floating point format.
- frequency: Frequency of input signal.
- wind_size: window size for processing in milliseconds.
- Overlap: Defines the amount of overlap in between two successive windows.
- N0: in case of LPC projects , it refers to number of LPC coefficients you want to calculate .(In case of median filtering of speech it refers to number of samples over which median is to be calculated.)

The input data is read into a linked list with root pointing to the root node of the linked list. The output is stored in (1-D or 2-D) linked list with root_output pointing to the root node of the linked list.

All the functions provided in C++ reference are available as projects. To run a specific project, ensure that SystemC library has been linked to your Microsoft Visual Studio 2008.

.

REFERENCES:

1) Digital Processing of Speech Signals , by Lawerence R. Rabiner and Ronald W. Schafer
2) Discrete time speech signal processing, by Thomas F. Quatieri
3) L.R.Rabiner and M.R. Sambeur, " An algorithm for Determining the endpoints of Isolated utterences," Bell syst. Tech.J. Vo.54, No.1, pp. 81-102, January 1975
4) L. R. Rabiner, M. J. Vheng, A.E. Rosenberg, and C.A. McGonegal, "A Comparitive performance study of several pitch detection algortihms," IEEE Trans. Acoust., Speech, and Signal Proc., Vol. ASSP-24, No. 5, pp. 399-418, October 1976.
5) H. Levitt, " Speech Processing Aids for the Deaf: An overview," IEEE trans. Audio and Electroacoutics, Vol. AU-21, pp 269-273, June 1973
6) B. Gold, " Computer Program for Pitch Extraction," J. Acoust. Soc. Am. Vol. 34, No.7, pp. 916-921, 1962
7) T.P. Barnwell, J.E. Brown, A.M. Bush, and C. R. Patisaul, "Pitch and Voicing in Speech Digitization," Res. Rept. No. E-21-620-74-B4-1, Georgia Inst. Of Tech., August, 1974.
8) M. J. Ross, H.L. Shaffer, A. Cohen, R.Freudberg, and H.J. Manly, " Average Magnitude Difference Function Pitch Estimator," IEEE Trans. Acoust., Speech and Signal Processing, Vol. ASSP-22, pp. 353-362, October 1974
9) M. M. Sondhi, " New Methods of Pitch Extraction," IEEE Trans. Audio and Electroacoustics, Vol. AU-16, No. 2, pp. 262-266, June 1968.
10) L. R. Rabiner, " On the use of Autocorrelation Analysis for Pitch Detection," IEEE Trans. Acoust. , Speech and Signal Proc., Vol. ASSP-25, No.1, pp 24-33, February, 1977.
11) J. J. Dubnowski, R.W. Schafer and L.R. Rabiner, "Real time Digital Hardware Pith Detector," IEEE Trans. Acoust., Speech and Signal Processing, Vol. ASSP-24, No.1, pp. 2-8, February 1976.
12) J. W. Tukey, " Nonlinear (Nonsuperposable) Methods for Smoothing Data," Congress Record, 1974 EASCON,p. 673,1974.
13) L. R. Rabiner, M. R. Sambur, and C. E. Schmidt, "Application of Non-linear Smoothing Algorithm to Speech Processing," IEE Trans. Acoust. Speech, and Signal Proc., Vol. ASSP-22, No. 1, pp.76-77, February 1974.
14) J.D. Markel and A.H. Gray, Jr., Linear Prediction of Speech, Springer- Verlag, New York, 1976.
15) J. Makhoul, "Linear Prediction: A tutorial Review," Proc. IEEE, Vol.63, pp.561-580.1975.
16) Steven F. Boll, "Suppression of Acoustic Noise in Speech Using Spectral Subtraction," IEEE Trans, on Acoustics, Speech, And Signal Processing, Vol. ASSP-27,No.2, April,1979.
17) Robert J. Mcaulay, " Speech Enhancement Using a Soft-Decision Noise Suppression Filter," IEEE Trans. On Acoustics, Speech, and Signal Processing, Vol. ASSP-28, No.2, April 1980

18) Yariv Ephraim, " Speech Enhancement Using a Minimum Mean Square Error Short Time Spectral Amplitude Estimator," IEEE Trans. On Acoustics, Speech and Signal Processing , Vol. ASSP-32, No.6, December 1984.
19) B.T. Lilly and K.K. Paliwal, " Robust Speech Recognition using Singular Value Decomposition based Speech Enhancement,".
20) Seehyum Kim, " Fixed-Point Optimization Utility for C and C++ based Digital Signal Processing Programs," IEEE Trans. On Circuits and Systems-II ; Analog and Digital Signal Processing, Vol.45, No. 11, November 1998.
21) Numerical Mathematics by: Matheus Grasselli and Dmitry Pelinovsky
22) Applied Numerical Methods for Engineering by: Robert J. Schilling and Sandra L. Harris
23) Algorithm Design by: Michael T. Goodrich and Roberto Tamassia